S.Graïne

LE LANGAGE C

Un cours concis et efficace

Des travaux pratiques corrigés



Le langage C

Avec travaux pratiques corrigés

S. Graïne

Les éditions l'Abeille

Les éditions l'Abeille
Boulevard Stiti – Tizi Ouzou
(Entrée principale de la gare ferroviaire)

Tél : 072 – 18 – 86 – 68

ISBN 9961 – 723 – 26 – 0
Dépôt légal : 1978 – 2003

TABLE DES MATIERES

Chapitre 9 : Les structures et les fonctions 99

Chapitre 10 : Les listes chaînées

Chapitre 1 : Concepts de base du langage C

Structure d'un programme C

En C, le bloc composant les instructions du programme doit être placé entre accolades à la suite du nom de la fonction principale **main()**.

De plus, dans tout programme C, on utilise des fonctions de la bibliothèque du C. Ces fonctions sont contenues dans des fichiers entête dont l'extension est «.h ». Parmi ces fichiers, on peut citer à titre d'exemple :

- **stdio.h** : qui contient, entre autres, les fonctions d'entrées/sorties scanf() et printf() respectivement pour lire les données au clavier et afficher les résultats à l' écran.
- **conio.h** : qui contient, entre autres, les fonctions clrscr() et gotoxy() qui permettent respectivement d'effacer l' écran et de positionner le curseur à l'écran avant un affichage.

A chaque fois qu'on utilise une fonction, il faut inclure au début du programme (avec **#include**) le nom du fichier entête qui la contient. La structure d'un programme C se présente donc comme suit :

```
#include <stdio.h>
#include <conio.h>
void main()
{

}
```

Entre les deux accolades se trouve le programme, qui est un ensemble de déclarations et d'instructions. En C, toute déclaration ou instruction doit se terminer par un point virgule.

Les commentaires

On peut faire un commentaire à n'importe quel endroit du programme : avant le main(), dans le main(), en début de ligne ou à la suite d'une instruction.

Un commentaire peut comprendre plus qu'une ligne et doit toujours commencer par /* et se terminer par */.

Les mots clés du langage C

auto	break	case	char	inline
const	continue	default	do	while
double	else	enum	extern	volatile
float	for	goto	if	void
unsigned	long	register	return	int
short	signed	sizeof	static	
struct	switch	typedef	union	

Les opérateurs et les expressions

Les opérateurs arithmétiques

Symbole	Description
+	Addition
-	Soustraction
/	Division
*	Multiplication
%	Modulo

Remarque :
Le modulo est le reste de la division entière. Par exemple : 5%2 = 1, 5%3 = 2 et 9%5 = 4.

Les opérateurs relationnels

Symbole	Description
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal

Il existe aussi 2 opérateurs de comparaison : = = (identique à), et != (différent de).

Les opérateurs logiques

Symbole	Description
&&	ET logique
||	OU logique
!	NON logique

L'opérateur d'affectation

Le symbole d'égalité (=) est utilisé pour affecter (pour assigner) la valeur d'une expression à un identificateur.

Exemple :

```
x = 2 * y + 5 ;   /* x reçoit le résultat de l'expression : 2*y + 5 */
```

Les variables

Les types de variables

Il existe trois types de base pour les variables en C :

int : pour les nombres entiers
float : pour les nombres réels
char : pour les caractères (lettres, chiffres, signes, ...).

Chaque type permet de représenter des données sur un certain nombre d'octets. Ce nombre étant limité, la plage des valeurs représentée par chaque type est donc limitée.

Type	Taille	Plage des valeurs
int	2 octets	-32 768 à 32 767
char	1 octet	256 caractères
float	4 octets	1.2e-38 à 3.4e38

Il existe d'autres types permettant de représenter d'autres plages de valeurs. Lorsque la plage des valeurs est strictement positive, on utilise des variables non signées (unsigned).

Type	Taille	Plage des valeurs
unsigned short int	2 octets	0 à 65 535
short int	2 octets	-32 768 à 32 767
unsigned long int	4 octets	0 à 4 294 967 295
long int	4 octets	-2 147 483 648 à 2 147 483 647
unsigned int	2 octets	0 à 65 535
double	8 octets	2.2e-308 à 1.8e308

Remarque :
La taille de chaque type diffère d'un compilateur à l'autre et d'un ordinateur à l'autre.

Les identificateurs

Les identificateurs sont les noms donnés aux variables et aux fonctions d'un programme. Ils ne doivent jamais commencer par un chiffre, ni contenir un caractère spécial (accent, opérateurs arithmétiques, opérateurs relationnels, opérateurs logiques et caractères de ponctuation), ni contenir un espace (l'identificateur doit être en un seul mot). Le caractère souligné « _ » est considéré comme un caractère normal.

Exemples :
x, somme1, _xy, n1, TAB, nb_fiches sont des identificateurs corrects.
5x, no-identificateur, nb emp sont des identificateurs incorrects car le 1er commence par un chiffre, le second contient l'opérateur arithmétique moins (-), et le dernier contient un espace.

Attention ! En C, les majuscules et les minuscules ne sont pas équivalentes. Tous les identificateurs suivants sont donc différents : TAB, TAb, Tab, tAB, tab, ...

Déclaration de variables

Déclarer une variable consiste à indiquer son type et à lui donner un nom.

Exemple :

```
int x, y ;
unsigned short a, A, b ;
float  u, v ;
```

Une variable peut être initialisée en même temps qu'elle est déclarée.

Exemple :

```
int x = 5, y ;
unsigned short a, A, b = 0 ;
float  u, v = 2.50 ;
```

Dans cet exemple, x est initialisé à 5, b à 0 et v à 2.50.

Un type donné peut apparaître plusieurs fois. Il n'est donc pas obligatoire de déclarer toutes les variables de même type en même temps. Les déclarations suivantes sont équivalentes aux précédentes :

```
int x = 5 ;
unsigned short A, b = 0 ;
int y ;
float  v = 2.50 ;
float  u ;
unsigned short a ;
```

Typedef

Le mot clé **typedef** permet de créer des synonymes.

Exemple :

```
typedef unsigned short int   Ushort
Ushort  a = 5, b, c = 10 ;
```

Ushort est synonyme de **unsigned short int**. Les variables a, b et c déclarées de type **Ushort** sont donc de type **unsigned short int**.

Les fonctions d'entrées et sorties : scanf() et printf()

Les fonctions d'entrées/sorties scanf() et printf() sont contenues dans le fichier entête stdio.h. Il faut donc l'inclure au début de chaque programme utilisant ensemble ou séparément ces deux fonctions.

Les entrées : La fonction scanf()

Elle permet de saisir des valeurs numériques, des caractères simples et des chaînes de caractères.

Syntaxe

scanf ("chaîne de format", variable1, ..., variableN) ;

Remarque : Le symbole & est obligatoire dans le scanf() devant le nom de toute variable, sauf si elle est de type chaîne de caractères.

Chaîne de format

Type	**Code format**	**Interprétation**
int	%d	Un entier
char	%c	Un caractère
char	%s	Une chaîne de caractères
float	%f	Un réel

Lire la valeur d'une variable

Pour lire la valeur d'une variable x déclarée de type entier (int), on écrit :

scanf("%d", &x) ;

On peut lire plusieurs valeurs à la fois. Si les variables x, y, z, s et t sont déclarées comme suit : int x,y ; float z ; char s, t[20] ;
On peut alors les lire en une seule fois :

scanf("%d%d%f%c%s", &x, &y, &z, &s, t) ;

Les sorties : La fonction printf()

Elle permet d'écrire des données sur l'écran en provenance de l'ordinateur. Ces données peuvent être numériques, des caractères ou des chaînes de caractères.

Syntaxe

printf ("chaîne de format", variable1, ..., variableN) ;

Chaîne de format

Les codes de format sont identiques à ceux de la fonction scanf() excepté pour les réels.

Type	Code format	Interprétation
float	%f	Notation décimale
float	%e	Notation exponentielle

La notation décimale affiche la partie entière suivie de 6 chiffres après le point décimal.
La notation exponentielle affiche la partie entière sur 1 à 10 chiffres, plus 6 chiffres après le point décimal et 2 chiffres pour l'exposant.

```
float x = 15.123456 ;          /* notation décimale */
float y = 15.123456e+02        /* notation exponentielle */
```

Afficher la valeur d'une variable

Pour afficher la valeur d'une variable x déclarée de type entier (int), on écrit : printf("%d", x) ;

On peut afficher plusieurs valeurs à la fois :

Exemple :

```
int x,y ; float z ; char s, t[20] ;
printf("%d %d %f %c %s", x, y, z, s, t) ;
```

Dans la fonction printf(), vous devez laisser au minimum un espace entre les différents codes de format afin que les différentes valeurs ne soient pas collées à l'affichage comme suit : "%d %d %f %c %s".
On peut faire également une tabulation avec « \t » :

```
printf("%d\t%d\t%f\t%c\t%s", x, y, z, s, t) ;
```

Afficher un message

Pour afficher un message, il suffit de le mettre entre guillemets dans la fonction printf().

Exemple : printf("Ceci est un message") ;

Introduire un message dans l'affichage d'une valeur

On peut faire apparaître un message en même temps que des valeurs dans un seul printf().

Exemple 1 : int x = 5, y = 7 ; /* déclarations */
printf("x = %d et y = %d", x, y) ;
L'instruction printf() va afficher : x = 5 et y = 7.

Exemple 2 : int x = 5, y = 7, z ; /* déclarations */
z = x + y ;
printf("%d + %d = %d", x, y, z) ;
L'instruction printf() va afficher : 5 + 7 = 12

Afficher la valeur d'une expression

On peut effectuer le calcul d'une expression arithmétique directement dans une fonction printf().

Exemple : int x = 5, y = 7 ;
printf("%d + %d = %d", x, y, x+y) ;
L'instruction printf() va afficher : 5 + 7 = 12

Pour changer de ligne

Le « \n » utilisé dans une fonction printf() permet d'effectuer un retour au début d'une nouvelle ligne avant d'afficher.

Exemple 1 :

```
int x = 5, y = 7 ;
printf("x = %d", x);
printf("y = %d", y);
```

Exemple 2 :

```
int x = 5, y = 7 ;
printf("x = %d", x);
printf("\ny = %d", y);
```

Dans l'exemple 1, on obtient l'affichage suivant : x = 5y = 7 (les 2 affichages sont collés), alors que dans l'exemple 2, on obtient :
x = 5
y = 7 (un affichage par ligne).

Le « \n » peut être utilisé au début, au milieu ou à la fin de la chaîne de format.

Exemple : int x = 5 ; char y = 'A', z[20]= "Chaîne";

```
printf("\nx = %d\ny = %c\nz = %s", x, y, z) ;
```

L'instruction printf() va afficher :

```
x = 5
y = A
z = Chaîne
```

Réserver le nombre d'espaces pour l'affichage

On peut réserver un nombre d'espaces pour la variable à afficher. Dans ce cas, il faut l'indiquer entre le symbole « % » et le code du format. Le signe « – » placé entre le « % » et le nombre permet de justifier l'affichage à gauche. Sinon, l'affichage est justifié à droite.

Exemple :

```
char    chaine1[10]="Message",
chaine2[15]="Messagelong";
int  x = 15 ;
float y = 123.50;
printf("%-15s%-15s", chaine1, chaine2);
printf("\n%5d%9.2f", x, y);
```

Le 1er printf() va afficher ce qui suit (le symbole « ^ » désigne un espace) :

Message^^^^^^^^Messagelong

Le 2nd printf() va afficher : ^^^15^^^123.50

%-15s : affiche sur 15 positions avec justification à gauche.
%5d : affiche sur 5 positions avec justification à droite.
%9.2f : affiche sur 9 positions incluant le point décimal et 2 chiffres après le point décimal, avec justification à droite.

Autres codes de format

Un préfixe peut être ajouté devant certains codes de format.

Préfixe	**Type**
h	short int ou unsigned short int
l	long int ou unsigned long int ou double
L	long int

Exemple :

```
short int a ; double b ;
unsigned short int c ; long int d ;
unsigned long int e ;
```

```
scanf("%hd%lf", &a, &b) ;
scanf("%hd%Ld", &c, &d) ;
scanf("%ld",&e) ;
printf("\n%hd %lf %hd", a, b, c) ;
printf(" %Ld %ld", d, e) ;
```

Exemple de programme

Programme C qui demande d'entrer deux nombres entiers, calcule leur somme, puis affiche le résultat sous la forme : Somme = …….

```
#include <stdio.h>
void main()  {
int x, y, somme ;
printf("Entrez 2 nombres entiers : ");
scanf("%d%d", &x, &y);
somme = x + y ;
printf("\n Somme = %d", somme);
}
```

Les fonctions gets() et puts()

La fonction gets() permet de lire une chaîne de caractères contenant des espaces (une phrase). Elle ne peut lire qu'une chaîne à la fois.
La fonction puts() permet d'afficher une chaîne de caractères à la fois.
Ces deux fonctions sont contenues dans le fichier entête « stdio.h ».

Syntaxe :

```
gets(chaine) ;
puts(chaine) ;
```

On peut également lire une chaîne de caractères contenant des espaces avec la fonction scanf() comme suit : `scanf("%[^\n]", chaine) ;`

Exemple :

```
#include <stdio.h>
void main()
{
char chaine1[20], chaine2[10];
printf("\n Entrez une phrase : ");
```

```
gets(chaine1) ;
printf("\n Entrez une phrase : ");
scanf("%[^\n]", chaine2) ;
printf("\n Voici les 2 phrases saisies : ") ;
printf("\n") ;
puts(chaine1) ;
printf("\n") ;
puts(chaine2);
}
```

Les constantes

Comme les variables, les constantes sont des espaces de stockage de données. Mais contrairement aux variables, les constantes ne changent pas de valeur tout au long du programme. Une constante doit être initialisée au moment de sa déclaration.

Constantes symboliques

Il existe deux façons de définir les constantes symboliques :

1) Avec le « #define » avant le main() (une constante par ligne) :

```
#define x 7
#define y 'A'
#define z "Vendredi"
```

2) Avec le mot clé « const » :

```
const int x = 7 ;
const char y = 'A' ;
const char z = "Vendredi" ;
```

x est une constante numérique, y est constante caractère et z est une constante chaîne de caractères.

Exemple :

```
#include <stdio.h>
#define x 500
void main() {
```

```
const char chaine[20]= "Prix";
printf("%s = %d D.A", chaine, x);
}
```

A l'exécution, le programme affiche : Prix = 500 D.A

Constantes énumérées

Le mot clé « enum » permet de créer d'autres types, puis de définir des variables de ces types.

```
enum Couleur { vert, noir, blanc } ;   /* on crée le type Couleur */
Couleur c1, c2 ; /* on déclare 2 variables de type Couleur */
```

Les variables c1 et c2 sont de type couleur et ont chacune trois valeurs possibles : vert, noir et blanc.
Chaque constante énumérée possède une valeur entière par défaut. Si aucune valeur n'est spécifiée explicitement, la première constante va avoir la valeur 0, la seconde va avoir la valeur 1 et ainsi de suite. Si des valeurs sont spécifiées : enum couleur { vert=5, noir, blanc=9 } ;

Dans ce cas, vert, noir et blanc vont avoir respectivement les valeurs 5, 6, et 9.

Quelques fonctions utiles du C

Les fonctions décrites dans le tableau ci-dessous sont toutes contenues dans le fichier entête « conio.h ».

Fonction	**Description**
clrscr()	Effacer l'écran.
getch()	Marquer une pause lors de l'exécution du programme jusqu'à ce qu'on appuie sur une touche.
getche()	Lire une variable caractère.
gotoxy(a, b)	Positionner le curseur à la colonne a et à la ligne b avant une opération de lecture ou d'écriture.

Remarque : L'instruction : caractere = getche() ;
est équivalente à : scanf("%c", &caractere) ;

TP N° 1

Exercice 1

Ecrire un programme C qui demande d'entrer un nombre entier, calcule son double et son triple, puis affiche le résultat comme suit :

Le double de .... = ...
Le triple de .... = ...

Exercice 2

Ecrire un programme C qui demande d'entrer le nom d'un étudiant et les 3 notes obtenues au cours de programmation. Le programme affiche ensuite la moyenne de cet étudiant comme suit :

La moyenne de ...... est : ....

Corrigé du TP N° 1

Exercice 1

```
#include <stdio.h>
void main()
{
int x ;
printf("Entrez un nombre entier : ");
scanf("%d", &x);
printf("\n Le double de %d = %d", x, 2*x);
printf("\n Le triple de %d = %d", x, 3*x);
}
```

Exercice 2

```
#include <stdio.h>
void main()
{
char nom[20] ; float n1, n2, n3, moy ;
printf("Nom de l'étudiant ? ");
scanf("%s", nom);
printf("\nSes 3 notes au cours de programmation ? ");
scanf("%f%f%f", &n1, &n2, &n3);
```

```
moy = (n1+n2+n3)/3 ;
printf("\n La moyenne de %s est : %.2f", nom, moy);
}
```

Chapitre 2 : Les structures de contrôle

L'instruction if ... else

L'instruction if .. else permet d'effectuer un test, et d'exécuter ensuite l'une des deux alternatives possibles, selon que la condition du test est vérifiée ou non.

Syntaxe :

```
if (condition)    {
                  bloc d'instructions ;
                  }
else    {
        bloc d'instructions ;
        }
```

Si le bloc d'instructions est composé d'une seule instruction, les accolades ne sont pas obligatoires.

Exemple 1 :

```
#include <stdio.h>
void main()
{
 int x;
printf("Entrez un nombre entier : ");
scanf("%d", &x);
if(x < 0)
        printf("\n Entier négatif");
else
        printf("\n Entier positif ou nul");
}
```

Exemple 2 :

```
#include <stdio.h>
void main()
{
enum jour { Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi};
jour j;    int no ;
printf("Entrez un No de jour entre 0 et 6 : ");
```

```
scanf("%d", &no);
j = jour(no);
if(j==Jeudi || j==Vendredi)
        printf("\n C'est un jour férié ...");
else
        printf("\n C'est un jour de semaine ...");
}
```

Si le numéro saisi est 5 ou 6, le programme affiche :
C'est un jour férié ...
Si le numéro saisi est entre 0 et 4, il affiche :
C'est un jour de semaine ...

Si le choix doit se faire entre plus que deux conditions, l'alternative va s'exprimer avec des « else if (condition) ».

Exemple :

```
#include <stdio.h>
void main()
{
 int x;
printf("Entrez un nombre entier : ");
scanf("%d", &x);
if(x < 0)
        printf("\n Entier négatif");
else if (x>0)
        printf("\n Entier positif") ;
else
        printf("\n Entier nul");
}
```

L'instruction while

Syntaxe: `while (condition)`

```
{
bloc d'instructions ;
}
```

Le bloc d'instructions mentionné (ou l'instruction) est répété tant que la condition est vraie (vraie signifie une valeur non nulle, et faux correspond à la valeur zéro).

Si le bloc d'instructions est composé d'une seule instruction, les accolades ne sont pas obligatoires.

Exemple : Imprimer les chiffres de 0 à 9.

```
#include <stdio.h>
void main ()
{
int i = 0 ;
while (i <= 9)
 {
   printf("\n %d", i) ;
   ++i ;  /* ou bien : i = i + 1 ;  */
 }
}
```

L'instruction do ... while

Syntaxe :
```
do   {
     bloc d'instructions
     } while (condition) ;
```

Le bloc d'instructions (ou l'instruction) est exécuté au moins une fois car le test de la condition ne s'effectue qu'au terme du 1er passage dans la boucle.

Si le bloc d'instructions est composé d'une seule instruction, les accolades ne sont pas obligatoires.

Exemple : Imprimer les chiffres de 0 à 9.

```
#include <stdio.h>
void main ()
{
int i = 0 ;
do
      {
      printf("\n %d", i) ;
      ++i ;   /* ou  i = i + 1 ; */
      } while (i <= 9) ;
}
```

L'instruction for

Syntaxe : for (valeur initiale ; condition ; pas)

```
                {
                bloc d'instructions ;
                }
```

Où « valeur initiale » désigne la valeur de début de la boucle, « condition » désigne la condition d'arrêt de la boucle, et « pas » désigne le nombre avec lequel on augmente ou on diminue la valeur initiale (le compteur).
Le bloc d'instructions mentionné (ou l'instruction) est répété tant que la condition est vraie. Si le bloc d'instructions est composé d'une seule instruction, les accolades ne sont pas obligatoires.

Exemple : Imprimer les chiffres de 0 à 9.

```
#include <stdio.h>
void main ()
{
int i;
for (i = 0 ; i <= 9 ; ++i)    /* ici, le pas = 1 */
        printf("\n %d", i) ;
}
```

Les boucles imbriquées

Les boucles peuvent être imbriquées les unes dans les autres. Une boucle for peut contenir une boucle for, une boucle while ou une boucle do...while et vice versa.

Exemple :

```
#include <stdio.h>
void main ()
{
int i, j, s;
for (i = 0 ; i < 2 ; ++i)
        for (j = 1 ; j <= 2 ; ++j)
                { s = i + j ;  printf("\n s = %d", s) ;
                }
}
```

Trace du programme ci-dessus :

i	j	s	Sortie à l'écran	Interprétation
0	1	1	s = 1	
0	2	2	s = 2	
0	3	2		La condition de la boucle j est fausse
1	1	2	s = 2	
1	2	3	s = 3	
1	3	3		La condition de la boucle j est fausse
2	3	3		La condition de la boucle i est fausse

L'instruction break

L'instruction break est utilisée pour mettre fin à une boucle ou sortir d'une instruction de sélection. Elle peut figurer dans les 4 instructions : while, do … while, for et switch … case.
Elle s'écrit tout simplement : break ;

Exemple :
Programme C qui permet de calculer la somme d'un ensemble de valeurs entières dont le nombre est inconnu. A la question « Autre valeur ? o/n » posée par le programme, si vous répondez 'o' (oui), la boucle while se poursuit. Si vous répondez 'n' (non), l'instruction «break » est exécutée et la boucle s'arrête. Le programme affiche alors le nombre et la somme des nombres saisis.

```
#include <stdio.h>
#include <conio.h>
void main ()
{
int val = 1, i = 0, s = 0, x ;
char rep ;

while(val) /* Condition toujours vraie car val = 1 */
        {
        printf("\n Entrez une valeur entière :") ;
       scanf("%d", &x) ;
       s = s + x ;
       ++i ;
```

```
        printf("\n Autre valeur ? o/n ") ;
         rep = getche() ;
        if(rep ==  'n' ) break ;
        }
printf("\n La somme des %d nombres = %d", i, s) ;
}
```

L'instruction continue

L'instruction continue permet de suspendre les instructions restantes dans une boucle (while, do … while, for) et de reprendre le traitement au début de la boucle.
Elle s'écrit tout simplement : continue ;

Exemple :
Calcul de la somme de 5 entiers positifs saisis par l'utilisateur.

```
#include <stdio.h>
void main ()
{
int i = 1, s = 0, x ;

do {
        printf("\n Entrez une valeur : ") ;
        scanf("%d", &x) ;

        if (x<0)
                {
                printf("\n La valeur doit être positive ") ;
                continue ; /* retour au début de la boucle */
                }

        /* Si x est positif, on l'ajoute dans la somme */
        s = s + x ;
        ++i ;
        } while (i <= 5) ;

printf("\n Somme = %d ", s) ;
}
```

L'instruction switch ... case

L'instruction switch ... case permet de sélectionner un groupe précis d'instructions parmi plusieurs.

Syntaxe :

```
switch (valeur ou expression)
        {
        case 1 : ........ ;
                 break ;
        case 2 : ........ ;
                 break ;
        ........................
        default : ....... ;
        }
```

Chaque « case » doit se terminer par une instruction « break » pour éviter que le programme n'exécute le « case » suivant.

Exemple :
Programme C qui reçoit une valeur entière et qui calcule selon le choix de l'utilisateur : 1- son double, 2- son triple ou 3- son carré. Si le choix de l'utilisateur n'est pas compris entre 1 et 3, le programme affiche, par défaut, le message « Choix inexistant ».

```
#include <stdio.h>
void main ()
{
int x, r, choix ;
printf("\n Entrez une valeur : ") ;   scanf("%d", &x) ;
printf("\n Choix ? 1-Double 2-Triple 3-Carré : ") ;  scanf("%d", &choix) ;
switch (choix)
        {
        case 1 : r = x * 2 ; printf("\n Double = %d", r) ;
                 break ;
        case 2 : r = x * 3 ; printf("\n Triple = %d", r) ;
                 break ;
        case 3 : r = x * x ; printf("\n Carré = %d", r) ;
                 break ;
        default : printf("\n Choix inexistant ") ;
        }
}
```

TP N° 2

Exercice 1

Ecrire un programme C qui permet de calculer les racines de l'équation du second degré suivante : $ax^2 - bx + c = 0$, sachant que :

a) Si $b^2 - 4ac > 0$, l'équation possède 2 racines x1 et x2 telles que :

$$x1 = (-b - \sqrt{b^2-4ac}\,) / 2a$$

$$x2 = (-b + \sqrt{b^2-4ac}\,) / 2a$$

b) Si $b^2 - 4ac = 0$, l'équation possède 1 racine $x = -b / 2a$.

c) Si $b^2 - 4ac < 0$, l'équation ne possède pas de racine.

Dans chaque cas, le programme devra afficher le message correspondant et la (ou les) racine(s) éventuelle(s).

NB : La fonction sqrt(x) calcule la racine carrée de x, et la fonction pow(x, n) calcule x à la puissance n. Ces deux fonctions sont contenues dans le fichier entête « math.h ».

Exercice 2

Ecrire un programme C qui calcule le prix d'un billet de cinéma selon une grille tarifaire qui comprend plusieurs classes. On accorde une escompte sur ce prix aux conditions suivantes :

- Si le client est âgé de 15 ans ou moins, ou âgé de 60 ans ou plus :
  - Si le billet est vendu un jour de semaine on accorde une escompte de 25%.
  - Sinon, on accorde une escompte de 10 % .
- Si le billet est vendu à un client qui n'entre pas dans cette catégorie d'âge :
  - Si le billet est vendu un lundi ou un jeudi, on accorde une escompte de 15%.
  - Sinon, on n'accorde aucune escompte.

Pour calculer le prix du billet, le programme demande d'entrer le jour sous forme d'un numéro (0. Samedi, 1. Dimanche, ..., 6. Vendredi), l'année en cours, l'année de naissance du client et le tarif de base. Sur le

billet, on veut imprimer l'âge du client, le montant de l'escompte accordée, ainsi que le prix du billet.

Exercice 3

Ecrire un programme C qui permet de calculer la somme suivante :

$$1 + 2 + \ldots + n,$$

n étant un entier positif entré par l'utilisateur à la demande du programme. Le programme ne doit pas accepter des valeurs de n inférieures à 1.

Exercice 4

Ecrire un programme C qui demande d'entrer un No entre 1 et 7, et donne le nom du jour correspondant (Samedi, Dimanche, ..., Vendredi) en utilisant un switch ... case.

Prévoir également un message d'erreur dans le cas où le numéro entré n'est pas compris entre 1 et 7.

Corrigé du TP N° 2

Exercice 1

```
#include <stdio.h>
#include <math.h>
void main()
{
float a, b, c, d, x, x1, x2 ;
printf("Les valeurs a, b et c de l'équation ? ");
scanf("%f%f%f", &a, &b, &c);
d = pow(b, 2) - 4 * a * c;
if( d  > 0)
  {
  x1 = (-b - sqrt(d))/(2*a) ;
  x2 = (-b + sqrt(d))/(2*a) ;
  printf("\n L'équation possède 2 racines : ");
  printf("x1 = %.2f et x2 = %.2f", x1, x2);
  }
else if( d == 0)
  {
```

```
  x = -b /(2*a) ;
  printf("\n L'équation possède une racine : ");
  printf("x = %.2f", x);
  }
else
  printf("\n L'équation ne possède pas de racine");
}
```

Exercice 2

```
#include <stdio.h>
#include <conio.h>
void main()
{
enum JOURS { Samedi, Dimanche, Lundi, Mardi,
                                        Mercredi, Jeudi, Vendredi } ;
JOURS jour ;
int AnneeCours, AnneeNaiss, age,  No ;
float escompte, PrixBase, prix ;
printf(" Année en cours ? ");
scanf("%d", &AnneeCours);
printf("\n Année de naissance ? ");
scanf("%d", &AnneeNaiss);
age = AnneeCours - AnneeNaiss ;
printf("\n Prix de base ? ");
scanf("%f", &PrixBase);
printf("\n No du jour : 0.Samedi 1. Dimanche, . . . ?");
scanf("%d", &No);
jour = JOURS(No) ;
if(age <= 15 || age >= 60)
  {
  if(jour >= Samedi && jour <= Mercredi)
         escompte = 0.25 * PrixBase ;
  else   escompte = 0.10 * PrixBase ;
  }
else
  {
  if(jour == Lundi || jour == Jeudi)
         escompte = 0.15 * PrixBase ;
  else escompte = 0 ;
  }
prix = PrixBase - escompte ;
```

```
/* Affichage */
clrscr() ;
printf("   Age : %d", age);
printf("\n Escomte : %.2f", escompte);
printf("\n Prix : %.2f", prix);
}
```

Exercice 3

```
#include <stdio.h>
#include <conio.h>
void main()
{
int n, i, somme = 0 ;

do
 {
  clrscr() ;
  printf(" Valeur de n ? ");
  scanf("%d", &n);
  if (n < 1)
        {
          printf("\n Erreur, la valeur de n doit être >= 1 ");
          getch() ;
        }
  } while(n < 1) ;

for(i=1 ; i<=n ; ++i)
        somme = somme + i ;
printf("\n Somme = %d", somme);
}
```

Exercice 4

```
#include <stdio.h>
void main()
{
int jour ;
printf("\n No du jour : 1.Samedi 2. Dimanche, . . . ?");
scanf("%d", &jour);
switch(jour)
        {
```

```
        case 1 : printf("\n Samedi "); break ;
        case 2 : printf("\n Dimanche "); break ;
        case 3 : printf("\n Lundi "); break ;
        case 4 : printf("\n Mardi "); break ;
        case 5 : printf("\n Mercredi "); break ;
        case 6 : printf("\n Jeudi "); break ;
        case 7 : printf("\n Vendredi "); break ;
        default : printf("\n No de jour inexistant ");
        }
}
```

Chapitre 3 : Les fonctions

Définition

Une fonction est un tout homogène destiné à une tâche bien définie (sous-programme).

Les fonctions permettent de répondre à deux objectifs principaux :

- Eviter les répétitions, et
- créer des modules accomplissant chacun une tâche spécifique.

Ces deux objectifs permettent de rendre les programmes plus lisibles et plus faciles à maintenir (à corriger et à améliorer).

Chaque fonction possède son nom propre qui permet de l'identifier. Au cours de son exécution, lorsque le programme rencontre le nom d'une fonction (appel de fonction), il effectue un saut vers l'adresse où se trouve la fonction en mémoire, exécute toutes les instructions qui la composent, puis revient à l'instruction qui suit immédiatement l'appel pour continuer le reste du programme.

```
void main()
{
. . .
fonction () ;   /* Appel de la fonction */
instruction ;  /* Instruction exécutée après l'appel */
. . .
} /* fin du main */
```

Remarque :
Un programme peut appeler plusieurs fonctions à des endroits différents de son exécution. Et une fonction peut elle même appeler d'autres fonctions.

Déclaration, appel et définition d'une fonction

Déclaration d'une fonction

Toute fonction utilisée doit être déclarée avant le main(). La déclaration indique au compilateur le nom de la fonction, son type et ses paramètres

(ou arguments). Elle doit toujours se terminer par un point virgule. Le type d'une fonction est déterminé par le type de la variable ou de l'expression que la fonction retourne. Si la fonction ne retourne aucune valeur, elle est de type « void ». La déclaration d'une fonction est appelée son prototype.

Syntaxe de la déclaration :

```
<Type fonction>  <nom fonction>(<type 1>,....., <type N>) ;
```

Où <type 1>,....., <type N> désignent respectivement le type du 1[er] paramètre, ..., le type du N[éme] paramètre de la fonction.

Exemple 1 : int fct1(int, int, float) ;
La fonction fct1() reçoit en paramètres 2 entiers et un réel. Elle retourne un entier : elle est donc déclarée de type « int ».

Exemple 2 : float fct2(void) ;
La fonction fct2() ne reçoit aucun paramètre (parenthèses vides ou void). Elle retourne un réel : elle est donc déclarée de type « float ».

Exemple 3 : void fct3(int, char) ;
La fonction fct3() reçoit en paramètres un entier et un caractère. Elle ne retourne rien : elle est donc déclarée de type « void ».

Exemple 4 : void fct4() ;
La fonction fct4() ne reçoit aucun paramètre (parenthèses vides ou void). Elle ne retourne rien : elle est donc déclarée de type « void ».

Appel d'une fonction

L'appel d'une fonction se fait dans le main() ou dans une autre fonction en mentionnant son nom et ses paramètres entre parenthèses. Si la fonction ne possède pas de paramètres, les parenthèses sont vides.

Il faut différencier deux cas :
1[er] cas : Si la fonction est de type « void » (ne retourne rien). Dans ce cas, la syntaxe de l'appel est :

```
<nom fonction>(<paramètre 1>,....., <paramètre N>) ;
```

Où <paramètre 1>,....., <paramètre N> désignent les noms des paramètres.

Exemple 1 : fct1(a,b,c) ;
Appel de la fonction fct1() de type « void » et ayant 3 paramètres.

Exemple 2 : fct2() ;
Appel de la fonction fct2() de type « void » et sans paramètres.

2ème cas : Si la fonction est d'un autre type que « void » (retourne un résultat). Dans ce cas, la syntaxe de l'appel est :

```
<variable> = <nom fonction>(<paramètre 1>,....., <paramètre N>) ;
```

Où <variable> est une variable de même type que la fonction appelée. Cette variable est déclarée dans le main() ou dans la fonction appelante. Son rôle est de recevoir le résultat retourné par la fonction.

Exemple :

```
int fct1(int) ; /* déclarations */
float fct2() ;
void main()
{
int x, y ; float z ;
. . .
x = fct1(y) ; /* appel de fct1() */
z = fct2() ; /* appel de fct2() : fonction sans paramètres */
. . .
}
```

Dans le cas où la fonction retourne un résultat (n'est pas de type « void »), l'appel de la fonction peut se faire directement dans une fonction printf() :

```
printf("%..", <nom fonction>(<paramètre 1>,....., <paramètre N>));
```

ou bien : `printf("%..",<nom fonction> ( ) );`

Le résultat retourné par la fonction est donc affiché directement.

Exemple :

```
float fct1(char) ;
int fct2() ;
void main() { char x ;
. . .
printf("\n %f", fct1(x) ) ;
printf("\n %d", fct2() ) ;
. . .
}
```

Définition d'une fonction

Définir une fonction consiste à écrire les instructions qui la composent. En général, la définition de la fonction se fait après le main().

Syntaxe de la définition :

```
<type fonction> <nom fonction>(<type1>  p1,....., <typeN>  pN)
  {
   .............. ;
   .............. ;  instructions de la fonction
  }
```

Où <type1>, ..., <typeN> et p1, ..., pN désignent respectivement les types et les noms des paramètres dans la fonction.

Exemple :
Programme qui contient une fonction appelée « somme ». Cette fonction reçoit 3 entiers en paramètres, calcule leur somme, puis retourne le résultat au main() qui l'affiche.

```
#include <stdio.h>
int somme(int, int, int) ;    /* déclaration */
void main ()
{
int  x, y , z, s ;
printf(" \n Entrez 3 nombres entiers  : ") ;
scanf("%d%d%d", &x, &y, &z);
s = somme (x, y, z) ;    /* appel de la fonction */
printf(" \n Somme =  %d", s) ;
}
/* Définition de la fonction */
int  somme(int x, int y, int z)
{
int d ;
d =  x + y + z ;
return  d ;
}
```

La fonction « somme » reçoit les valeurs de x, y et z, les additionne et place le résultat dans d. Celui-ci est retourné à la fonction main() qui le récupère dans s, puis l'affiche.

Les variables locales

En plus de pouvoir passer des paramètres à une fonction, celle-ci peut avoir ses propres variables qui sont déclarées lors de sa définition. Les variables déclarées dans le main() sont locales au main(), et les variables déclarées dans une fonction sont locales à cette fonction, même si elles portent le même nom. La portée des variables est locale à la fonction qui les contient.

Exemple :

```
#include <stdio.h>
void fct() ;
void main ()
{
int i  = 5 ;
printf("\n Avant l'appel de fct() : i = %d", i);
fct() ;
printf("\n Après l'appel de fct() : i = %d", i) ;
}
void fct()
{
int i = 3 ;
i = i + 5 ;
printf(" \n Dans fct() : i  =  %d", i) ;
}
```

Le programme affiche ce qui suit :

> Avant l'appel de fct() : i = 5
> Dans fct() : i = 8
> Après l'appel de fct() : i = 5

Dans le main(), i vaut 5, alors qu'il vaut 8 dans fct(). Ce qui prouve que ce n'est pas le même i. Le compilateur alloue deux espaces mémoires différents au i déclaré dans le main() et au i déclaré dans la fonction fct().

De plus, en C, on peut déclarer des variables n'importe où dans le programme et n'importe où dans une fonction, pas seulement au début. On peut également faire des déclarations de variables dans des blocs limités par des accolades, et ces variables sont locales aux blocs dans lesquelles elles sont déclarées.

Exemple :

```
#include <stdio.h>
void fct() ;
void main ()
{
int i  = 5 ;
printf(" \n Avant l'appel de fct() : i  =  %d", i)  ;
fct() ;
printf(" \n Après l'appel de fct() : i  =  %d", i)  ;
}

void fct( )
{
int i = 7 ;
printf("\n Dans fct()  avant le bloc : i = %d", i);

  {
   printf("\n Au début du bloc : i  = %d", i) ;
   int i = 11 ;
   printf("\n A la fin du bloc : i  = %d", i);
  }

printf("\n Dans fct()  après le bloc : i = %d", i);
}
```

Le programme affiche ce qui suit :

Avant l'appel de fct() : i = 5
Dans fct() avant le bloc : i = 7
Au début du bloc : i = 7
A la fin du bloc : i = 11
Dans fct() après le bloc : i = 7
Après l'appel de fct() : i = 5

Les variables globales

Une variable est dite globale, si elle est connue de toutes les fonctions qui sont compilées au sein du même programme source. Pour qu'elle soit ainsi, il faut la déclarer avant le main().

Exemple :

```
#include <stdio.h>
void fct() ;
int i ;  /* variable globale */
void main ()
{
i = 5 ;
printf("\n Avant l'appel de fct() : i  =  %d" , i)  ;
fct() ;
printf("\n Après l'appel de fct() : i  =  %d",  i)  ;
}
void fct()
{
i = i + 5 ;
printf(" \n Dans fct() : i  =  %d",  i)  ;
}
```

Le programme affiche ce qui suit :

> Avant l'appel de fct() : i = 5
> Dans fct() : i = 10
> Après l'appel de fct() : i = 10

La variable i est déclarée globale. Dès qu'elle subit une modification, celle-ci apparaît dans le reste du programme. C'est la même variable partout.

Remarque :
On déclare une variable globale lorsque la donnée qu'elle contient est utilisée par un ensemble de fonctions et qu'on ne souhaite pas passer cette donnée comme paramètre d'une fonction à une autre. Toutefois, ceci peut constituer un danger, car une même donnée manipulée par plusieurs fonctions peut être source d'erreurs difficiles à localiser.

Passage de paramètres à une fonction

Les paramètres passés à une fonction peuvent être de types différents. De plus, toute expression valide en C peut être passée comme paramètre à une fonction, incluant les constantes, les expressions mathématiques et logiques, et les fonctions retournant une valeur.

Les paramètres sont des variables locales

Les paramètres d'une fonction sont des variables locales de cette fonction. Ils sont déclarés dans l'entête de la fonction (entre parenthèses).

Exemple :

```
#include <stdio.h>
void fct(int ) ;
void main ()
{
int i  = 5 ;
printf("\n Avant l'appel de fct() : i  =  %d", i)  ;
fct(i) ;
printf("\n Après l'appel de fct() : i  =  %d", i)  ;
}
void fct(int x) /* La variable x est locale à fct() */
{
x = x + 5 ;
printf(" \n Dans fct() : x  =  %d",  x)  ;
}
```

Le programme affiche ce qui suit :

```
Avant l'appel de fct() : i = 5
Dans fct() : x = 10
Après l'appel de fct() : i = 5
```

A l'appel de fct(), la valeur de i (égale à 5) est copiée dans x (variable locale de fct()). Celle-ci l'augmente de 5, puis l'affiche (ce qui donne x = 10 dans fct()). Au retour dans le main(), i vaut toujours 5.

Le paramètre étant une variable locale, même si x s'appelait i (même nom que dans le main()), le programme sera identique.

```
void fct(int i) /* La variable i est locale à fct() */
{
i = i + 5 ;
printf(" \n Dans fct() : i  =  %d",  i)  ;
}
```

A l'exécution le programme affichera ce qui suit :

Avant l'appel de fct() : i = 5
Dans fct() : i = 10
Après l'appel de fct() : i = 5

Utilisation de fonctions comme paramètres d'autres fonctions

Une fonction qui retourne une valeur peut être utilisée comme paramètre d'une autre fonction.

Exemple :

```
#include <stdio.h>
int Double (int ) ;
int  Triple (int) ;
void main ()
{
int x, y ;
printf("Donnez une valeur entière : ") ;
scanf("%d", &x) ;
y = Double (Triple(x)) ;
printf("\nLe double du triple de %d = %d", x, y);
}
int Double(int v )
{
return (2*v) ;
}
int Triple(int v )
{
return (3*v) ;
}
```

Attention ! Le mot « double » (avec d minuscule) est un mot clé. C'est pour cela que nous avons écrit « Double » (avec D majuscule).

A l'exécution le programme affiche ce qui suit :

Donnez une valeur entière : 5
Le double du triple de 5 = 30

Le programme appelle d'abord la fonction triple(5) qui retourne la valeur 15, et appelle ensuite la fonction double(15), qui retourne la valeur 30 au main() qui l'affiche.

Les paramètres par défaut

A l'appel d'une fonction, celle-ci doit recevoir un nombre de paramètres égal au nombre de paramètres déclarés dans son prototype. De plus, ces paramètres doivent également être de mêmes types que ceux déclarés. Si ces deux conditions ne sont pas respectées, le compilateur signalera des erreurs lors de la phase de compilation du programme.
Il existe une seule exception pour laquelle le nombre de paramètres à l'appel d'une fonction peut être inférieur au nombre de paramètres déclarés : c'est lorsqu'on donne des valeurs par défaut aux paramètres.

Exemple : `long fonction( int x, int y = 10) ;`
Cette déclaration signifie que la fonction est de type « long » et qu'elle reçoit deux entiers. Lorsqu'on appellera cette fonction, on peut lui passer juste un entier, le second prendra automatiquement la valeur 10, par défaut.

N'importe quel paramètre peut avoir une valeur par défaut. Toutefois, si un paramètre possède une valeur par défaut, tous ceux déclarés après, dans l'entête de la fonction, doivent en avoir une.

Exemples de déclarations correctes :

```
int fct1( int,  int, int a=10) ;
void fct2(char, long, int x=2, int y=65) ;
int fct3(char c1='A', char c2='B', int c3=10) ;
```

Exemples de déclarations incorrectes :

```
int fct1( int, int a=10, int ) ;
void fct2(char, long, int x=2, int y) ;
int fct3(char c1='A', char c2='B', int) ;
```

Exemple :

```
#include <stdio.h>
int perimetre (int, int larg = 5 ) ;
void main ()
{
int x = 30, y = 20, z = 10 ;
printf("\n 1er  périmètre   =  %d", perimetre(x, y));
printf("\n  2éme périmètre   =  %d", perimetre(x));
```

```
printf("\n  3éme périmètre   =  %d", perimetre(z));
}
int perimetre(int a, int b )
{ return ( 2*(a+b) ) ;
}
```

A l'exécution, le programme affiche :

1er périmètre = 100
2éme périmètre = 70
3éme périmètre = 30

Pour le 1er périmètre : Longueur = 30 et largeur = 20.
Pour le 2éme périmètre : Longueur = 30 et largeur = 5 (par défaut).
Pour le 3éme périmètre : Longueur = 10 et largeur = 5 (par défaut).

Surcharge des fonctions

Qu'est-ce que la surcharge de fonctions ?

En C, on peut créer plusieurs fonctions avec le même nom. Ceci est appelé : surcharge des fonctions ou polymorphisme. Les fonctions doivent différer dans la liste de leurs paramètres : soit dans le nombre de paramètres, soit dans le type d'au moins un des paramètres, ou soit tout ça à la fois.

Exemple :

```
int fonction(int, int) ;
int fonction(int, float) ;
int fonction(int) ;
```

Dans l'exemple ci-dessus, si la fonction appelée reçoit en paramètres :

- deux entiers : c'est la première fonction qui est exécutée ;
- un entier et un réel : c'est la deuxième fonction qui est exécutée ;
- un entier : c'est la troisième fonction qui est exécutée.

Les fonctions surchargées peuvent également avoir des types différents. La liste des paramètres doit cependant être différente, car la déclaration de deux fonctions avec le même nom, la même liste des paramètres et des types différents va générer une erreur de compilation.

En d'autres termes, la surcharge de fonctions consiste à déclarer des fonctions :

- avec le même nom,
- des types identiques ou différents, et
- une liste de paramètres obligatoirement différente.

Exemple :
```
int fonction(int, int) ;
void fonction(int, int) ; /* erreur */
```

Dans cet exemple, la 2ème déclaration va générer une erreur de compilation.

Utilité de la surcharge de fonctions

Supposons qu'on souhaite écrire une fonction qui calcule le double de la valeur entrée par l'utilisateur. On veut être capable de lui passer un « int », un « long », un « float » ou un « double ». Sans la surcharge des fonctions, on doit créer quatre fonctions avec quatre noms différents :

```
int Double1(int) ;
long Double2(long) ;
float Double3(float) ;
double Double4(double) ;
```

Par contre, avec la possibilité de la surcharge de fonctions, un seul nom de fonction suffit.

Exemple :
```
#include <stdio.h>
int Double(int) ;
long Double(long) ;
float Double(float) ;
double Double(double) ;
void main ()
{
int a = 6500 ;  long b = 65000 ;
float c = 11.5 ; double d = 6.5e20 ;
printf("\n  Le double de %d  = %d ", a, Double(a)) ;
printf("\n  Le double de  %Ld = %Ld ", b, Double(b)) ;
printf("\n  Le double de  %f = %f ", c, Double(c)) ;
printf("\n  Le double de  %lf = %lf ", d,Double(d)) ;
}
```

```
int Double(int valeur)  { return (2 * valeur); }
long Double(long valeur)  { return (2 * valeur); }
float Double(float valeur) { return (2 * valeur); }
double Double(double valeur)  { return (2 * valeur); }
```

A l'exécution, le programme affiche :

Le double de 6500 = 13000
Le double de 65000 = 130000
Le double de 11.5 = 23
Le double de 6.5e20 = 13e21

A l'appel de la fonction Double(), le compilateur exécute entre les 4 fonctions ayant le même nom, celle dont le paramètre est de même type que la variable reçue en paramètre.

Les fonctions en ligne

Lorsqu'une fonction est appelée lors de l'exécution d'un programme, celui-ci effectue un saut pour aller l'exécuter. A la fin de celle-ci, il revient à l'instruction suivant l'appel. Si la fonction est appelée 15 fois, le programme effectue un saut vers cette fonction 15 fois, car il en existe juste une seule copie.

Lorsqu'une fonction est déclarée en ligne (précédée du mot clé inline), le compilateur effectue une copie de cette fonction à l'endroit de chaque appel, lors de la phase de compilation. Si la fonction est appelée, par exemple, 15 fois dans le programme, il va en exister donc, 15 copies à la fin de la phase de compilation. Lors de la phase d'exécution, le programme n'effectue aucun saut vers cette fonction.

Cette façon de faire permet de réduire le temps d'exécution du programme (pas de sauts vers la fonction). Toutefois, ceci entraîne une augmentation de l'espace mémoire occupé par le programme (plusieurs copies d'une même fonction).

Que faire donc ? Une fonction sera déclarée en ligne (inline) à deux conditions simultanées :

- elle est petite (4 ou 5 instructions), et
- elle est appelée plusieurs fois dans le programme.

Exemple : inline int fonction(int, int) ;

Les fonctions récursives

On dit qu'une fonction est récursive lorsqu'elle s'appelle elle même de façon répétitive jusqu'à ce qu'une condition donnée soit vérifiée.

Les fonctions récursives sont utilisées lorsqu'on effectue des calculs de types récursifs.

Exemple :
Soit la fonction factorielle (dont la notation mathématique est : !) définie comme suit : $n ! = 1 \times 2 \times \ldots\ldots \times n$, sachant que : $0 ! = 1 ! = 1$.

Pour avoir le résultat de n!, il faut calculer (n-1)!, puis le multiplier par n. Et pour calculer (n-1)!, il faut calculer (n-2)!, puis le multiplier par (n-1). Et ainsi de suite. D'une façon générale : $n ! = n \times (n-1) !$ (calcul récursif).

Le programme contenant une telle fonction s'écrira donc comme suit :

```
#include <stdio.h>
int factorielle(int) ;
void main ()
{
int n, resultat ;
printf("Donnez la valeur de n ") ;
scanf("%d", &n) ;
resultat = factorielle(n) ;
printf("\n  %d! = %d ", n, resultat) ;
}
int factorielle(int n)
{
if (n ==0 || n == 1) return (1) ;
else return (n * factorielle(n-1)) ;
}
```

La fonction factorielle(n) va s'appeler elle même avec la valeur (n-1). La fonction factorielle(n-1) va à son tour, s'appeler elle même avec la valeur (n-2). Et ainsi de suite jusqu'à n égal à 1. Dans ce cas, factorielle(1) vaut 1,

et l'appel récursif s'interrompt. Le compilateur va effectuer donc tous les autres calculs laissés en suspend.

TP N° 3

Exercice 1

Ecrire un programme C qui demande d'entrer 2 entiers, puis calcule le plus grand des 2 en utilisant une fonction. Le résultat doit être affiché comme suit : Le plus grand entre .... et .... est : ......
La fonction reçoit en paramètre les 2 entiers et renvoie le plus grand des deux au main() qui l'affiche.

Exercice 2

Ecrire un programme C qui demande d'entrer le nom d'un étudiant ainsi que ses trois notes, puis calcule sa moyenne, sa plus grande note et sa plus petite note.

La moyenne, la plus grande note et la plus petite note sont calculées chacune par une fonction. Chacune de ces fonctions reçoit en paramètres les 3 notes, et retourne le résultat correspondant (la moyenne, la plus grande, ou la plus petite des 3 notes, selon le cas).

Le programme affichera le nom de l'étudiant, sa moyenne, sa plus grande note et sa plus petite note.

Exercice 3

Soit n un entier positif et P un polynôme défini comme suit :

P1 = 1 (ou P = 1, si n = 1)
P2 = 1 (ou P = 1, si n = 2)
Pn = Pn-1 + Pn-2.

Ecrire un programme C contenant une fonction récursive permettant de calculer les polynômes de ce genre. La fonction doit recevoir la valeur de n en paramètre et retourner le résultat au programme principal qui l'affiche sous la forme suivante : P(n) = ...

Exercice 4

Soit n un entier positif ou nul et S un polynôme défini comme suit :

S0 = 2
S1 = 3
S2 = 4, et
Sn = 2(Sn-1 +5) + Sn-2.

Ecrire un programme C contenant une fonction récursive permettant de calculer les polynômes de ce genre. La fonction doit recevoir la valeur de n en paramètre et retourner le résultat au programme principal qui l'affiche sous la forme suivante : S(n) = ……

Corrigé du TP N° 3

Exercice 1

```
#include <stdio.h>
int PlusGrand(int, int) ;
void main()
{
int nb1, nb2, pg ;
printf("Donnez 2 entiers : ");
scanf("%d%d", &nb1, &nb2);
pg = PlusGrand(nb1, nb2) ;
printf("\nLe plus grand entre %d et %d est : %d", nb1, nb2, pg);
}
int PlusGrand(int nb1, int nb2)
{
if(nb1 > nb2) return nb1 ;
else return nb2 ;
}
```

Exercice 2

```
#include <stdio.h>
#include <conio.h>
float moyenne(float, float, float) ;
float PlusGrande(float, float, float) ;
```

```
float PlusPetite(float, float, float) ;
void main()
{
char nom[20];
float note1, note2, note3, moy, pg, pp ;
printf("Nom de l'étudiant ? ");
scanf("%s", nom);
printf("\n Ses 3 notes ? ");
scanf("%f%f%f", &note1, &note2, &note3);

moy = moyenne(note1, note2, note3) ;
pg = PlusGrande(note1, note2, note3) ;
pp = PlusPetite(note1, note2, note3) ;

/* Affichage */
clrscr() ;
printf("\n Nom : %s", nom) ;
printf("\n Moyenne = %.2f", moy) ;
printf("\n Plus grande note = %.2f", pg) ;
printf("\n Plus petite note = %.2f", pp) ;
}
float moyenne(float n1, float n2, float n3)
{
return (n1+n2+n3)/3 ;
}
float PlusGrande(float n1, float n2, float n3)
{
if(n1 >= n2 && n1 >= n3) return n1 ;
else if(n2 >= n1 && n2 >= n3) return n2 ;
else return n3 ;
}
float PlusPetite(float n1, float n2, float n3)
{
if(n1 <= n2 && n1 <= n3) return n1 ;
else if(n2 <= n1 && n2 <= n3) return n2 ;
else return n3 ;
}
```

Exercice 3

```
#include <stdio.h>
int polynome(int) ;
```

```
void main()
{
int n, R ;
printf("Valeur de n >= 1 ? ");
scanf("%d", &n);
R = polynome(n) ;
printf("\n P(%d) = %d", n, R);
}
int polynome(int n)
{
if(n==1 || n==2) return 1 ;
else return ( polynome(n-1) + polynome(n-2) ) ;
}
```

Exercice 4

```
#include <stdio.h>
int polynome(int) ;
void main()
{
int n, R ;
printf("Valeur de n >= 0 ? ");
scanf("%d", &n);
R = polynome(n) ;
printf("\n S(%d) = %d", n, R);
}
int polynome(int n)
{
if(n==0) return 2 ;
else if(n==1) return 3 ;
else if(n==2) return 4 ;
else return ( 2*(polynome(n-1)+5) + polynome(n-2) );
}
```

Chapitre 4 : Les tableaux

Définition

Un tableau est un ensemble de données ayant le même type.
Il peut s'agir d'un tableau d'entiers, d'un tableau de réels, d'un tableau de caractères, ou d'un tableau de structures (cf. chapitre 6).

Tableaux à une dimension

Syntaxe de déclaration : <type> <nom tableau>[dimension] ;

Où <type> désigne le type des données du tableau, et « dimension » désigne le nombre de données que peut contenir ce tableau.

Exemple :

```
int tab1[10] ;
float tab2[15] ;
char tab3[20] ;
```

tab1 est un tableau de 10 entiers, tab2 est un tableau de 15 réels et tab3 est un tableau de 20 caractères.

Manipulation d'un tableau à une dimension

Chaque élément d'un tableau à une dimension est identifié par la position à laquelle il se trouve dans le tableau. En langage C, le 1er élément d'un tableau se trouve à la position 0, le 2nd à la position 1, ..., et le dernier à la position (dimension-1).

La position est également appelée indice : le 1er élément a pour indice 0, le 2nd a pour indice 1, ..., et le dernier a pour indice (dimension-1).

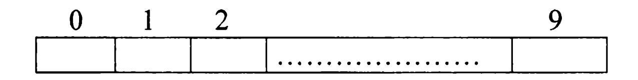


Tableau de dimension 10

Exemple :
Programme C qui initialise un tableau de 20 entiers à zéro, et l'affiche sous la forme suivante :

nom_tableau [0] = 0
....
nom_tableau [19] = 0

```
#include <stdio.h>
void main ()
{
int tab[20], i ;
/* Initialisation du tableau à zéro */
for(i=0 ; i<20 ; ++i)
        tab[i]=0 ;
/* Affichage du tableau */
for(i=0 ; i<20 ; ++i)
        printf("\n tab[%d] = %d ", i, tab[i]) ;
}
```

Tableaux à deux dimensions

Syntaxe de déclaration :

```
<type>  <nom tableau>[<dimension 1>] [<dimension 2>] ;
```

Où <dimension 1> et <dimension 2> désignent respectivement le nombre de lignes et le nombre de colonnes du tableau.

Exemple :

```
int tab1 [10] [5];
float tab2 [15][10] ;
```

tab1 est un tableau d'entiers ayant 10 lignes et 5 colonnes. Ce tableau peut contenir 50 entiers.
tab2 est un tableau de réels ayant 15 lignes et 10 colonnes. Ce tableau peut contenir 150 réels.

Manipulation d'un tableau à deux dimensions

Chaque élément d'un tableau à deux dimensions est identifié par l'indice de la ligne et l'indice de la colonne sur lesquelles il se trouve dans le tableau.

En langage C, la 1ere ligne d'un tableau a pour indice 0, la 2nde a pour indice 1, ..., et la dernière a pour indice (<dimension 1> – 1). Et la 1ere colonne a pour indice 0, la 2nde a pour indice 1, ..., et la dernière a pour indice (<dimension 2> – 1).

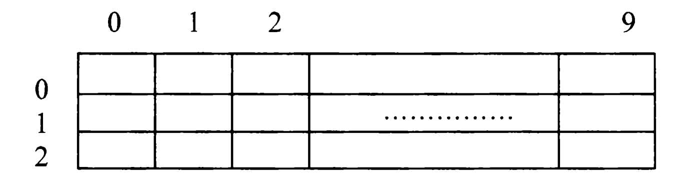


Tableau de 3 lignes et 10 colonnes

Exemple :
Programme C qui initialise un tableau d'entiers de 20 lignes, 15 colonnes à zéro, et l'affiche sous la forme suivante :

nom_tableau [0] [0] = 0
....
nom_tableau [19][14] = 0

```
#include <stdio.h>
void main ()
{
int tab[20][15], i, j ;

/* Initialisation du tableau à zéro */
for(i=0 ; i<20 ; ++i)
  for(j=0 ; j<15 ; ++j)
    tab[i][j] = 0 ;

/* Affichage du tableau */
for(i=0 ; i<20 ; ++i)
  for(j=0 ; j<15 ; ++j)
    printf("\n tab[%d][%d] = %d ", i, j, tab[i][j]);
}
```

Tableaux à plusieurs dimensions

On peut déclarer des tableaux de dimension supérieure à 2.

Syntaxe de déclaration :

<type> <nom tableau>[<dimension 1>] ..... [<dimension N>] ;

Exemple :

```
int tab1 [3][10][5];
char tab2 [5][7][20] ;
```

Le principe de manipulation d'un tableau à plusieurs dimensions est le même que pour un tableau à deux dimensions. Si on utilise des boucles imbriquées pour parcourir le tableau, la $1^{ére}$ boucle correspondra à <dimension 1>, la 2nde à <dimension 2>, ..., et $N^{éme}$ à <dimension N>.

Exemple :

Programme C qui initialise un tableau d'entiers de dimensions 10, 20 et 30 à zéro, et l'affiche sous la forme suivante :

nom_tableau[0] [0] [0]= 0

....

nom_tableau[9][19][29] = 0

```
#include <stdio.h>
void main ()
{
int tab[10][20][30], i, j, k ;

/* Initialisation du tableau à zéro */
for(i=0 ; i<10 ; ++i)
  for(j=0 ; j<20 ; ++j)
    for(k=0 ; k<30 ; ++k)
      tab[i][j][k] = 0 ;

/* Affichage du tableau */
for(i=0 ; i<10 ; ++i)
  for(j=0 ; j<20 ; ++j)
    for(k=0 ; k<30 ; ++k)
      printf("\n tab[%d][%d][%d] = %d ", i, j, k, tab[i][j][k]) ;
}
```

TP N° 4

Exercice 1

Ecrire un programme C qui calcule la moyenne des éléments d'un tableau de réels de dimension 10.

Les éléments du tableau sont entrés comme données à la demande du programme.

Le résultat est affiché comme suit : La moyenne = ...

Exercice 2

Soit un tableau de réels de dimension 10. Ecrire un programme qui calcule le plus grand élément du tableau ainsi que la position à laquelle il se trouve.

Le résultat est affiché comme suit :
Le plus grand élément est : <nom tableau>[position] = ...

Exercice 3

Soit un tableau de réels de 5 lignes et 4 colonnes. Ecrire un programme qui calcule le plus grand élément du tableau ainsi que la position à laquelle il se trouve.

Le résultat est affiché comme suit :
Le plus grand élément est : <nom tableau>[ligne][colonne] = ...

Exercice 4

Ecrire un programme C qui permet de saisir un tableau de 10 réels, puis l'affiche trié par ordre croissant.

Exercice 5

Ecrire un programme C qui permet de saisir un tableau de 10 réels, puis l'affiche trié par ordre décroissant.

Corrigé du TP N° 4

Exercice 1

```
#include <stdio.h>
void main()
{
float tab[10], som=0, moy ;
int i ;
for(i=0 ; i<10 ; ++i)
        {
        printf("\n Entrez un réel : ");
        scanf("%f", &tab[i]);
        som = som + tab[i] ;
        }
moy = som/10 ;
printf("\nLa moyenne = %.2f", moy) ;
}
```

Exercice 2

```
#include <stdio.h>
void main()
{
float tab[10], PlusGrand ;
int i , position;
for(i=0 ; i<10 ; ++i)
        {
        printf("\nEntrez un réel : ");  scanf("%f", &tab[i]);
        }
/* Recherche du plus grand élément et sa position */
PlusGrand = tab[0] ; position = 0 ;
for(i=1 ; i<10 ; ++i)
        if(tab[i] > PlusGrand)
                {
                PlusGrand = tab[i] ; position = i ;
                }
/* Affichage du résultat */
printf("\nLe plus grand élémént est tab[%d] = %.2f",
                                        position, PlusGrand) ;
}
```

Exercice 3

```
#include <stdio.h>
void main()
{
float tab[5][4], PlusGrand ;
int i, j , ligne, colonne;
for(i=0 ; i<5 ; ++i)
        for(j=0 ; j<4 ; ++j)
          {
          printf("\nEntrez un réel : ");
          scanf("%f", &tab[i][j]);
          }
/* Recherche du plus grand élément et sa position */
PlusGrand = tab[0][0] ; ligne = 0 ; colonne = 0 ;
for(i=0 ; i<5 ; ++i)
        for(j=0 ; j<4 ; ++j)
          if(tab[i][j] > PlusGrand)
            {
              PlusGrand = tab[i][j] ;
               ligne = i ; colonne = j ;
            }
printf("\nLe plus grand élément est tab[%d][%d] =  %.2f",
                                        ligne, colonne, PlusGrand) ;
}
```

Exercice 4

```
#include <stdio.h>
#include <conio.h>
void main()
{
float tab[10], temp ;
int i, j ;
for(i=0 ; i<10 ; ++i)
        {
        printf("\nEntrez un réel : ");
        scanf("%f", &tab[i]);
        }
/* Tri par ordre croissant */
for(i=0 ; i<9 ; ++i)
        for(j=i+1 ; j<10 ; ++j)
```

```
		if(tab[i] > tab[j])
			{
			temp = tab[i] ;
			tab[i] = tab[j] ;
			tab[j] = temp ;
			}
printf("\nAffichage du tableau trié :") ;
clrscr() ;
for(i=0 ; i<10 ; ++i)
		printf("\n %.2f", tab[i]);
}
```

Exercice 5

Même programme que ci-dessus.
On remplace juste : `if(tab[i] > tab[j])`
Par : `if(tab[i] < tab[j])`

Chapitre 5 : Les chaînes de caractères

Une chaîne de caractères est un tableau de caractères. Il existe, toutefois, plusieurs fonctions dans la bibliothèque du C qui permettent des manipulations spécifiques aux tableaux de caractères (aux chaînes de caractères), telles que les fonctions de concaténation, de comparaison, etc. Nous en verrons quelques unes ci-dessous.

Manipulation d'un tableau de caractères

Contrairement à un tableau d'entiers ou de réels qui ne peut être manipulé qu'élément par élément, un tableau de caractères à une dimension peut être manipulé élément par élément et peut être également manipulé en un seul bloc.

Exemple : char chaine[20] ;

Cette chaîne peut être affiché, par exemple, caractère par caractère :

```
for(i=0 ; i<20 ; ++i)
        printf("%c ", chaine[i]) ;
```

comme elle peut être affiché en un seul bloc : printf("%s", chaine) ;

Un tableau de caractères à deux dimensions peut être manipulé élément par élément et peut être manipulé ligne par ligne. Il ne peut pas être manipulé en un seul bloc.

Exemple : char chaines[5][20] ;

Le tableau « chaines » pouvant contenir 5×20 = 100 caractères, ou 5 chaînes de 20 caractères chacune, peut être affiché, par exemple, caractère par caractère :

```
for(i=0 ; i<5 ; ++i)
  for(j=0 ; j<20 ; ++j)
     printf("%c ", chaines[i][j]) ;
```

comme il peut être affiché ligne par ligne (ou chaîne par chaîne) :

```
for(i=0 ; i<5 ; ++i)
   printf("%s", chaines[i]) ;
```

Le principe de manipulation d'un tableau de caractères à deux dimensions peut être généralisé à un tableau à plusieurs dimensions.

Les fonctions de concaténation de chaînes

Syntaxe : strcat (chaîne1, chaîne2) ;
strncat (chaîne1, chaîne2, nb) ;

strcat() recopie chaîne2 à la suite de chaîne1, et strncat() recopie les nb premiers caractères de chaîne2 à la suite de chaîne1. Ces deux fonctions sont contenues dans le fichier entête « string.h ».

Exemple :
Programme C qui permet de saisir deux chaînes de caractères, les concatène, puis les affiche.

```
#include <stdio.h>
#include <string.h>
void main ()
{
char c1[20], c2[20] ;
printf("Entrez deux chaînes de caractères : ") ; scanf("%s%s", c1, c2) ;
strcat(c1, c2) ; /* concaténation */
printf("\n Affichage des 2 chaînes : %s  %s", c1, c2) ;
}
```

A l'exécution, le programme affiche ce qui suit :

Entrez deux chaînes de caractères : bon jour
Affichage des deux chaînes : bonjour jour

Les fonctions de comparaison de chaînes

Syntaxe : strcmp (chaîne1, chaîne2) ;
strncmp (chaîne1, chaîne2, nb) ;

strcmp() compare chaîne1 et chaîne2 et retourne une valeur entière :

- positive, si chaîne1 est située après chaîne2 dans l'ordre alphabétique.
- négative, si chaîne1 est située avant chaîne2 dans l'ordre alphabétique.
- égale à 0, si chaîne1 et chaîne2 sont identiques.

strncmp() compare les nb premiers caractères de chaîne1 avec la chaîne2 et retourne une valeur entière qui s'interprète de la même façon que précédemment.
Ces deux fonctions sont contenues dans le fichier entête « string.h ».

Exemple :
Programme C qui permet de saisir deux chaînes de caractères, puis les affiche triées par ordre alphabétique.

```
#include <stdio.h>
#include <string.h>
void main ()
{
char c1[20], c2[20] ;
printf("Entrez deux chaînes de caractères : ") ;  scanf("%s%s", c1, c2) ;
if(strcmp(c1, c2) < 0) printf("\n %s  %s", c1, c2) ;
else printf("\n %s  %s", c2, c1) ;
}
```

Les fonctions de copie de chaînes

Syntaxe : strcpy (chaîne1, chaîne2) ;
strncpy (chaîne1, chaîne2, nb) ;

strcpy() recopie chaîne2 dans chaîne1 (c'est une affectation), et strncpy() recopie les nb premiers caractères de chaîne2 dans chaîne1. Ces deux fonctions sont contenues dans le fichier entête « string.h ».

Exemple :
Programme C qui permet de saisir une chaîne de caractères, la recopie dans une autre chaîne, puis les affiche.

```
#include <stdio.h>
#include <string.h>
void main ()
{
char c1[20], c2[20] ;
printf("Entrez une chaîne de caractères : ") ;
scanf("%s", c1) ;
strcpy(c2, c1) ; /* copie de c1 dans c2 */
printf("\n Voici les deux chaînes : %s  %s", c1, c2) ;
}
```

A l'exécution, le programme affiche ce qui suit :

```
Entrez une chaîne de caractères :  bonjour
Voici les deux chaînes : bonjour  bonjour
```

Calcul de la taille d'une chaîne de caractères

Pour calculer la taille d'une chaîne de caractères (nombre de caractères qui la composent), on utilise la fonction strlen() qui reçoit en paramètre le nom de la chaîne et retourne sa taille (un entier).

Syntaxe : t = strlen(chaîne) ;

Où t doit être déclaré de type entier. Cette fonction est contenue dans le fichier entête « string.h ».

Exemple :
Programme C qui permet de saisir une chaîne de caractères, puis affiche sa taille.

```
#include <stdio.h>
#include <string.h>
void main ()
{
char c[20] ;   int t ;
printf("Entrez une chaîne de caractères : ") ;  scanf("%s", c) ;
t = strlen(c) ; /* calcul de la taille de c */
printf("\n Taille de la chaîne saisie = %d", t) ;
}
```

A l'exécution, le programme affiche ce qui suit :

Entrez une chaîne de caractères : bonjour
Taille de la chaîne saisie = 7

Conversion de caractères en majuscules et en minuscules

Dans la bibliothèque du C, on dispose de deux fonctions tolower() et toupper() qui permettent respectivement de convertir un caractère en minuscule et de convertir un caractère en majuscule. Chacune de ces fonctions reçoit en paramètre un caractère (celui à convertir) et retourne un caractère (le résultat de la conversion).

Syntaxe :
```
x = toupper(c) ;
x = tolower(c) ;
```

Où x et c doivent être déclarés de type caractère. Ces deux fonctions sont contenues dans le fichier « ctype.h ».

Exemple :
Programme C qui permet de saisir une chaîne de caractères en minuscules, la convertit en majuscules, puis l'affiche.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void main ()
{
char c[20] ;    int t , i;
printf("Saisie en minuscules : ") ;
scanf("%s", c) ;
/* calcul de la taille de la chaîne */
t = strlen(c) ;
/* Conversion en majuscules caractère par caractère */
for(i=0 ; i<t ; ++i) c[i]= toupper(c[i]) ;
printf("\n Voici la chaîne en majuscules : %s", c) ;
}
```

Remarque :
Pour la conversion de majuscules à minuscules, il suffit de remplacer toupper() par tolower().

TP N° 5

Exercice 1

Ecrire un programme C qui demande d'entrer une liste de 10 noms, puis les affiche en majuscules.

Exercice 2

Ecrire un programme C qui demande d'entrer une liste de 10 noms, puis les affiche triés par ordre alphabétique.

Exercice 3

Ecrire un programme C permettant de saisir une phrase au clavier et d'en afficher le nombre de mots.

Corrigé du TP N° 5

Exercice 1

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
void main ()
{
char nom[10][20] ; /* 10 noms de 20 caractères */
int t , i, j;

/* Saisie des 10 noms et conversion en majuscules */
for(i=0 ; i<10 ; ++i)
        {
        printf("\n Saisie d'un nom en minuscules : ") ;
        scanf("%s", nom[i]) ;
        t = strlen(nom[i]) ;
        /* Conversion en majuscules */
        for(j=0 ; j<t ; ++j)
                nom[i][j]=toupper(nom[i][j]) ;
        }
```

```
/* Affichage */
clrscr() ;
for(i=0 ; i<10 ; ++i)
        printf("\n %s", nom[i]) ;
}
```

Exercice 2

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
void main ()
{
char nom[10][20] , temp[20];
int t , i, j;
/* Saisie des 10 noms */
for(i=0 ; i<10 ; ++i)
        {
        printf("\n Saisie d'un nom : ") ;
        scanf("%s", nom[i]) ;
        }
/*  Tri par ordre alphabétique */
for(i=0 ; i<9 ; ++i)
   for(j=i+1 ; j<10 ; ++j)
        if(strcmp(nom[i], nom[j]) > 0)
        {
        strcpy(temp, nom[i]) ;
        strcpy(nom[i], nom[j]) ;
        strcpy(nom[j], temp) ;
        }

/* Litse des noms triés par ordre alphabétique */
clrscr() ;
for(i=0 ; i<10 ; ++i)
        printf("\n %s", nom[i]) ;
}
```

Exercice 3

```
#include <stdio.h>
#include <string.h>
```

```
#include <ctype.h>
void main () { char phrase[80];   int i, nb, t;
/* Saisie de la phrase  */
printf("Entrez une phrase : \n");
gets(phrase);

/* On initialise le nombre d'espaces à zéro */
nb=0;

/* On calcule la taille de la phrase */
t = strlen(phrase);

/* On parcourt la phrase et on compte le nombre d'espaces */
for(i=0 ; i<t ; ++i)
        if(phrase[i]==' ')
                ++nb;

/* Le nombre de mots est égal au nombre d'espaces + 1 */
++nb;

/* On affiche le nombre de mots  */
printf("\n Le nombre de mots = %d", nb);
}
```

Chapitre 6 : Les structures

Structures et variables de type structure

Une structure est un ensemble d'informations constituant une entité ou un enregistrement.

Syntaxe de déclaration :

```
typedef   struct   <nom structure> {
                        <membre 1> ;
                        <membre 2> ;
                        ..............
                        <membre N> ;
                        };
```

Où <membre 1>, <membre 2>, …, <membre N> sont des noms de variables avec leurs types.

Exemple :

```
typedef struct compte
        {
        int NoCompte ;
        char nom[20], EtatCompte;
        float solde ;
        };
```

« NoCompte », « nom », « EtatCompte » et « solde » sont les variables membres de la structure « compte ».

Une fois la structure définie, on peut déclarer des variables du type de cette structure comme suit :

```
<nom structure>  variable ;
```

Exemple :

```
compte c1, c2 ;
```

c1 et c2 sont deux variables de type « compte ». Ce qui signifie que chacune de ces deux variables est composée d'un numéro de compte, d'un nom, d'un état de compte et d'un solde.

Manipulation des variables structures

La manipulation d'une variable structure se fait membre par membre. Et l'accès à un membre d'une structure se fait comme suit :

variable.membre

Où « variable » est la variable de type <nom structure>, et « membre » est un membre de la structure manipulée. Les deux étant séparés par un point (par exemple : c1.NoCompte).

Exemple :
Programme C qui permet de saisir une variable structure de type « compte », puis l'affiche.

```
#include <stdio.h>
typedef struct compte { int NoCompte; char nom[20]; };
void main()
{
compte c ;
printf("\n No de compte et le nom du client ?") ;
scanf("%d%s", &c.NoCompte, c.nom) ;
printf("\n %d \t %s", c.NoCompte, c.nom) ;
}
```

Tableau de structures

Syntaxe de déclaration : <nom structure> <nom tableau>[dimension] ;

Exemple : compte C[10] ;
«compte» est le nom de la structure, et «C» est le nom du tableau de structures de type « compte » de dimension 10.

L'accès à un membre d'un tableau de structure se fait comme suit :
<nom tableau>[i].membre

Où <nom tableau>[i] fait référence au $i^{éme}$ élément du tableau, et « membre » à un membre précis de la structure manipulée. Les deux étant séparés par un point.

Par exemple : C[i].NoCompte permet l'accès au « NoCompte » du $i^{éme}$ élément du tableau « C ».

Exemple :
Programme C qui permet de saisir un tableau de structures de type « compte », puis l'affiche.

```
#include <stdio.h>
#include <conio.h>
typedef struct compte
        { int NoCompte; char nom[20] ; };
void main()
{
compte c[10] ;  int i ;
for(i = 0 ; i < 10 ; ++i)
        {
        printf("\n No de compte et le nom client ? ") ;
        scanf("%d%s", &c[i].NoCompte, c[i].nom) ;
        }
/* Affichage */
clrscr() ;
for(i = 0 ; i < 10 ; ++i)
        printf("\n %d \t %s", c[i].NoCompte, c[i].nom) ;
}
```

Structure membre d'une autre structure

Une structure peut figurer parmi les membres d'une autre structure. Dans ce cas, elle doit être déclarée avant la structure qui la contient.

Exemple :

```
typedef struct date { int jour, mois, annee ; } ;
typedef struct compte
        {
         int NoCompte; char nom[20] ;
         date DtOuverture ;
        };
```

« DtOuverture » est une variable structure de type « date », membre de la structure « compte ». Dans ce cas, la structure « date » est déclarée obligatoirement avant la structure « compte ».

Lorsqu'un membre d'une structure est lui même une structure, on accède alors à ce membre comme suit :

<variable structure>.<variable sous_structure>.<membre>

Exemple : c.DtOuverture.jour

Où « c » est une variable de type « compte ».

Dans le cas d'un tableau, l'accès se fait comme suit :

<nom tableau>[indice].<variable sous_structure>.<membre>

Exemple : C[i].DtOuverture.jour

Où « C[i] » fait référence au $i^{éme}$ élément du tableau « C » de type « compte ».

Exemple 1 :

Programme C qui permet de saisir une variable structure, puis l'affiche. La variable structure utilisée possède une variable structure comme membre.

```
#include <stdio.h>
typedef struct date { int j, m, a ; } ;
typedef struct compte
  { int NoCpte; char nom[20]; date dt ;} ;
void main()
{
compte c ;
printf("No compte et nom du client ? ") ;
scanf("%d%s", &c.NoCpte, c.nom);
printf("\n Date d'ouverture du compte ? ") ;
scanf("%d%d%d", &c.dt.j, &c.dt.m, &c.dt.a);
/* Affichage */
printf("\nNo compte : %d\tNom : %s", c.NoCpte, c.nom);
printf("\nDate : %d/%d/%d", c.dt.j, c.dt.m, c.dt.a);
}
```

Exemple 2 :

Programme C qui permet de saisir un tableau de structures, puis l'affiche. La variable structure utilisée possède une variable structure comme membre.

```
#include <stdio.h>
typedef struct date { int j, m, a ; };
```

```
typedef struct compte
  { int NoCpte; char nom[20]; date dt; };
void main()
{
compte c[10] ;  int i ;
for(i = 0 ; i < 10 ; ++i)
 {
 printf("No compte et nom du client ? ") ;
 scanf("%d%s", &c[i].NoCpte, c[i].nom);
 printf("\n Date d'ouverture du compte ? ") ;
 scanf("%d%d%d", &c[i].dt.j, &c[i].dt.m, &c[i].dt.a);
 }
/* Affichage */
for(i = 0 ; i < 10 ; ++i)
 {
 printf("\nNo compte : %d", c[i].NoCpte);
 printf("\nNom client : %s", c[i].nom);
 printf("\nDate:%d/%d/%d", c[i].dt.j, c[i].dt.m, c[i].dt.a);
 }
}
```

Chaque structure est une entité à part. Il est par conséquent possible d'utiliser un même nom de membre pour représenter des données de types différents.

Exemple :

```
typedef struct premiere
          { float a ; int b ; char c };
typedef struct deuxieme
          { char a ; float b,c ; } ;
```

TP N° 6

Exercice 1

Soit la structure « client » définie comme suit :

```
typedef struct client     {
                char nom[20], prenom[20] , telephone[15];
                };
```

Ecrire un programme C qui permet de saisir et d'afficher un client.

Exercice 2

Soit une structure «etudiant » ayant pour membres : le nom, le prénom, la moyenne et l'adresse (No de rue, nom de rue, nom de ville et code postal) de l'étudiant.

Ecrire un programme C qui permet de saisir et d'afficher un tableau d'étudiants de dimension 10.

Exercice 3

Même énoncé que l'exercice 2. La liste des étudiants doit être affichée par ordre alphabétique.

Corrigé du TP N° 6

Exercice 1

```
#include <stdio.h>
#include <conio.h>
typedef struct client
        { char nom[20], prenom[20], telephone[15] ;
        };
void main ()
{
client C ;
printf("Nom, Prénom et téléphone du client ? ") ;
scanf("%s%s%s", C.nom, C.prenom, C.telephone) ;
/* Affichage */
clrscr() ;
printf("\n%s %s %s", C.nom, C.prenom, C.telephone) ;
}
```

Exercice 2

```
#include <stdio.h>
#include <conio.h>
typedef struct adresse
        {  int No ; char rue[20], ville[20], cp[10] ;
        };
```

```
typedef struct etudiant
        {
         char nom[20], prenom[20] ; float moyenne ;
         adresse adr ;
        };
void main ()
{
etudiant E[10] ;
int i ;
for(i=0 ; i<10 ; ++i)
 {
 printf("Nom, Prénom et moyenne de l'étudiant ? ") ;
 scanf("%s%s%f", E[i].nom, E[i].prenom, &E[i].moyenne) ;
 printf("\n Adresse (No rue, rue, ville et cp) ? ") ;
 scanf("%d%s%s%s", &E[i].adr.No, E[i].adr.rue,
                               E[i].adr.ville, E[i].adr.cp) ;
 }

/* Affichage */
clrscr() ;
for(i=0 ; i<10 ; ++i)
 {
 printf("\n%s %s %.2f", E[i].nom, E[i].prenom, E[i].moyenne) ;
 printf(" %d %s %s %s", E[i].adr.No, E[i].adr.rue, E[i].adr.ville,
                                                      E[i].adr.cp);
 }
}
```

Exercice 3

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct adresse
        {
         int No ; char rue[20], ville[20], cp[10] ;
        };
typedef struct etudiant
        {
         char nom[20], prenom[20] ; float moyenne ;
         adresse adr ;
        };
```

```
void main ()
{
etudiant E[10], temp ;
int i,j ;
for(i=0 ; i<10 ; ++i)
  {
  printf("Nom, Prénom et moyenne de l'étudiant ? ") ;
  scanf("%s%s%f", E[i].nom, E[i].prenom, &E[i].moyenne) ;
  printf("\n Adresse (No rue, rue, ville et cp) ? ") ;
  scanf("%d%s%s%s", &E[i].adr.No, E[i].adr.rue,
                              E[i].adr.ville, E[i].adr.cp) ;
  }

/* Tri par ordre alphabétique */
for(i=0 ; i<9 ; ++i)
        for(j=i+1 ; j<10 ; ++j)
                if(strcmp(E[i].nom,E[j].nom)>0)
                        {
                        temp=E[i];
                        E[i]=E[j];
                        E[j]=temp;
                        }
/* Affichage */
clrscr() ;
for(i=0 ; i<10 ; ++i)
        {
        printf("\n%s %s %.2f", E[i].nom, E[i].prenom, E[i].moyenne) ;
        printf(" %d %s %s %s", E[i].adr.No, E[i].adr.rue,
                                        E[i].adr.ville, E[i].adr.cp);
        }
}
```

Chapitre 7 : Les pointeurs

Définition

Un pointeur est une variable qui contient l'adresse d'une autre variable. Il possède le même type que cette variable. Il est déclaré précédé d'une étoile.

Exemple :

```
float *x ;
int *a, *b ;
char *t ;
```

x est un pointeur qui va contenir l'adresse d'un réel, a et b sont des pointeurs qui vont contenir chacun l'adresse d'un entier, et t est un pointeur qui va contenir l'adresse d'un caractère ou d'une chaîne de caractères.

Notations

```
int u, *v = &u ;
```

On déclare un entier u et un pointeur v qu'on initialise avec « &u ». La notation « &u » se lit «adresse de la variable u ». Elle désigne l'adresse de la place mémoire allouée par le compilateur à la variable u. Le symbole « & » est appelé opérateur d'adressage.

La déclaration précédente est équivalente à :

```
int u , *v ;
v = &u ;
```

Le pointeur v n'est pas initialisé en même temps qu'il est déclaré. Il est initialisé par la suite dans le programme. Remarquez qu'il n'y a pas d'étoile devant le v. Dans une instruction du programme, « *v » a une autre signification.

```
int u, *v = &u;
*v = 3 ;
```

Dans une instruction (pas dans une déclaration), « *v » permet d'accèder à la variable dont l'adresse est contenue dans v. Le symbole « * » est appelé

opérateur d'indirection. C'est un opérateur qui agit exclusivement sur une variable pointeur.
Puisque le pointeur v contient l'adresse de la variable u, *v signifie donc u.

Exemple 1 :

```
#include <stdio.h>
void main()
{
int a = 5, *b ;
b = &a ;    /* Le pointeur b reçoit l'adresse de a */
printf("a = %d    et    *b = %d", a, *b) ;
}
```

A l'exécution, le programme affiche ce qui suit :

a = 5   et   *b = 5

*b = 5, car le pointeur b contient l'adresse de a, et à cette adresse se trouve la valeur de a (égale à 5).

Exemple 2 :

```
#include <stdio.h>
void main()
{
int a = 3, *b = &a;
printf("\n a = %d", a) ;
*b = 7 ;
printf("\n a = %d", a) ;
}
```

A l'exécution, le programme affiche les résultats suivants :

a = 3
a = 7

Puisque le pointeur b contient l'adresse de a, l'instruction « *b = 7 » est équivalente à « a = 7 ».

Passage de paramètres à une fonction

Lorsqu'on passe un paramètre (un argument) par pointeur à une fonction, on dit qu'on fait un passage de paramètre par adresse. Contrairement au

passage par valeur, les modifications apportées à ces paramètres par la fonction en question seront les nouvelles valeurs de ces paramètres pour la suite du programme.

Passage par valeur

Dans le passage par valeur, la fonction reçoit la valeur de la variable passée en paramètre.

Exemple :
Programme C qui contient une fonction fct() qui reçoit un entier par valeur.

```
#include <stdio.h>
void fct(int) ;
void main()
{
int a = 5 ;
printf("\n Avant fct() : a = %d", a);
fct(a);    /* passage par valeur */
printf("\n Après fct() : a = %d", a);
}
void fct(int a)
{
a = a + 7 ;
printf("\n Dans fct() : a = %d", a);
}
```

Après exécution, le programme affiche les résultats suivants :

```
Avant fct() : a = 5
Dans fct() :  a = 12
Après fct() : a = 5
```

Le paramètre déclaré lors de la définition de la fonction est une variable locale à cette fonction (le compilateur lui alloue un autre espace mémoire). La variable a du main() et la variable a de la fonction fct(), quoique ayant le même nom, sont différentes, puisqu'elles occupent des espaces mémoires différents. C'est ce qui explique que a passe à 12 dans la fonction, mais il reste à 5 dans le main(), car les changements ont eu lieu dans un autre espace mémoire.

Passage par adresse

Dans le passage par adresse, la fonction ne reçoit pas la valeur de la variable passée en paramètre (comme ci-dessus), mais elle reçoit son adresse.

Exemple :
Programme C qui contient une fonction fct() qui reçoit un entier par adresse.

```
#include <stdio.h>
void fct(int *) ;
void main()
{
int a = 5 ;
printf("\n Avant fc() : a = %d", a);
fct(&a);  /* passage par adresse */
printf("\n Après fct() : a = %d", a);
}
void fct( int *x)
{
*x = *x + 7 ;
printf("\n Dans fct() : *x = %d", *x);
}
```

Après exécution, le programme affiche les résultats suivants :

> Avant fct() : a = 5
> Dans fct() : *x = 12
> Après fct() : a = 12

Le pointeur x de la fonction fct() reçoit l'adresse de la variable a du main(). Avec l'opérateur d'indirection placé en avant du x dans fct(), on va à l'adresse de a et on augmente son contenu de 7. Ce qui veut dire que le main() et la fonction fct() travaillent dans le même espace mémoire. C'est ce qui explique que la modification apportée à la variable *x dans fct() (en réalité à la variable a du main()) est la nouvelle valeur de a dans la suite du programme.

Utilités du passage par adresse

1°) Si une fonction reçoit un paramètre et que celui-ci subit des modifications dans la fonction (il prend une nouvelle valeur). Si on souhaite continuer la suite du programme avec cette nouvelle valeur, le passage de ce paramètre par adresse permet de récupérer directement la nouvelle valeur qu'il prend dans la fonction.

2°) Une fonction ne peut pas retourner plus qu'une valeur. Mais grâce au passage de paramètres par adresse, on peut récupérer plusieurs valeurs à la fois, même si la fonction est de type « void ».

Exemple :

```
#include <stdio.h>
void calculs(int, int *, int *, int *) ;
void main()
{
int x, Double, Triple, Carre ;
printf("Entrez un entier :");
scanf("%d", &x);
calculs(x, &Double, &Triple, &Carre);
printf("\n Double de %d = %d", x, Double);
printf("\n Triple de %d = %d", x, Triple);
printf("\n Carré de %d = %d", x, Carre);
}
void calculs(int x, int *Double, int *Triple, int *Carre)
{
*Double = 2 * x ;
/* contenu du contenu de Double reçoit 2*x */
*Triple = 3 * x ;
/* contenu du contenu de Triple reçoit 3*x */
*Carre = x * x ;
/* contenu du contenu de Carre reçoit x*x */
}
```

La fonction calculs() reçoit en paramètres un entier et trois adresses. L'entier va servir pour les calculs du double, du triple et du carré, et les adresses vont servir à recevoir les résultats de ces calculs.

3°) Lors d'un passage de structures à une fonction (voir chapitre 9), le passage par adresse permet de réduire le temps et la mémoire nécessaires à

l'exécution du programme. Dans le passage par valeur, lorsqu'une fonction reçoit une variable structure, celle-ci est copiée dans une autre variable structure de même type, locale à la fonction. Dans le passage par adresse, la fonction reçoit juste l'adresse de la variable structure. D'où l'économie de temps et de mémoire.

Passage d'un tableau à une fonction

Le passage d'un tableau de données comme paramètre à une fonction est toujours un passage par adresse, car le nom d'un tableau contient en réalité son adresse. Les modifications apportées par la fonction à ce tableau vont apparaître dans le reste du programme.

Syntaxe :

Déclaration :

```
<type>  <nom fonction>(<type>  <nom tableau>[ ]) ;
```

Appel :

```
<nom fonction>(<nom tableau>) ;
```

Définition :

```
<type>  <nom fonction>(<type>  <nom tableau>[ ])
{ ................................ }
```

Exemple :
Programme qui permet de saisir un tableau d'entiers, le trie par ordre croissant, puis l'affiche.

Le tri se fait dans une fonction qui reçoit en paramètre le tableau saisi. L'affichage se fait dans le main().

```
#include <stdio.h>
void tri_croissant(int t[]) ;
void main (void)
{
int t[5], i ;
for(i=0 ; i<5 ; ++i)
	{
	printf("\n Donnez t[%d] : ", i) ;
	scanf("%d", &t[i]) ;
	}
tri_croissant(t) ;  /* appel de la fonction */
```

```
printf("\n Tableau trié : ") ;
for(i=0 ; i<5 ; ++i)
        printf("\n t[%d] = %d ", i, t[i]) ;
}
void tri_croissant(int t[])
{
int i, j, temp ;
for(i=0 ; i<4 ; ++i)
        for(j=i+1 ; j<5 ; ++j)
          if(t[i] > t[j])
                { temp=t[i] ; t[i]=t[j] ; t[j]=temp ; }
}
```

Retour d'un tableau par une fonction

Comme pour le passage d'un tableau à une fonction, le retour d'un tableau par une fonction est toujours un retour par adresse.

Syntaxe : Déclaration : <type> * <nom fonction>(paramètres) ;
Appel : <nom fonction>(paramètres) ;
Définition : <type> * <nom fonction>(paramètres)
{.....................................}

Où <type> désigne le type du pointeur retourné par la fonction. Il est le même que celui du tableau.

Lorsqu'une fonction se termine, le lien entre le reste du programme et les variables locales de cette fonction est coupé. Pour maintenir le lien avec l'emplacement mémoire contenant le tableau de données (celui dont on retourne l'adresse), il faut faire une allocation de mémoire en utilisant la fonction malloc() contenue dans le fichier entête « alloc.h ». Le lien avec cet espace mémoire sera maintenu jusqu'à ce qu'on le libère avec la fonction free() contenue dans le même fichier entête.
La fonction malloc() reçoit en paramètre la taille (en octets) de l'espace mémoire à allouer, et retourne l'adresse de cet espace.

Exemple :
Programme qui permet de saisir un tableau d'entiers dans une fonction, le retourne au main() qui l'affiche.

```
#include <stdio.h>
#include <alloc.h>
int * saisie() ;
void main ()
{
int *t, i ;
t = saisie() ;
printf("\n Tableau saisi : ") ;
for(i=0 ; i<5 ; ++i)
        printf("\n t[%d] = %d ", i, t[i]) ;
free(t) ;
}
int * saisie()
{
int i, *t=(int*)malloc(5*sizeof(int)) ;
for(i=0 ; i<5 ; ++i)
        {
        printf("\n Donnez t[%d] : ", i) ;  scanf("%d", &t[i]) ;
        }
return t ;
}
```

Dans la fonction saisie(), on alloue un espace mémoire pour 5 entiers avec la fonction malloc(). L'adresse de cet espace mémoire (de ce tableau) est affectée au pointeur t. Ce pointeur t qui est en même temps le nom du tableau (puisque le nom d'un tableau contient en réalité son adresse) est retourné à la fin de la saisie. Ce pointeur est récupéré dans un autre pointeur t du main() (on peut l'appeler autrement, ça ne changera rien), puis on affiche les données qui s'y trouvent.

TP N° 7

Exercice 1

Que va afficher le programme suivant :

```
#include <stdio.h>
void main()
{ int u = 5, v, *pu, *pv ;
pu = &u ;  pv = &v ;
```

```
v = *pu + 2 ;
printf("\n  u = %d ", u) ;
printf("\n *pu = %d ", *pu) ;
printf("\n v = %d ", v) ;
printf("\n *pv = %d ", *pv) ;
}
```

Exercice 2

Que va afficher le programme suivant :

```
#include <stdio.h>
void main()
{ int i, j = 10;
int *pi, *pj = &j ;
*pj = j + 7 ; i = *pj + 3 ;
pi = pj ; *pi = i + j ;
printf("\n  i = %d ", i) ;
printf("\n  j = %d ", j) ;
printf("\n  *pj = %d ", *pj) ;
printf("\n  *pi = %d ", *pi) ;
printf("\n  *pi + 3 = %d ", (*pi + 3) ) ;
}
```

Exercice 3

Que va afficher le programme suivant :

```
#include <stdio.h>
void fct1(int, int) ;
void fct2(int *, int*) ;
void main()
{ int u = 1, v = 3;
printf("\n u = %d          v = %d ", u, v) ;
fct1(u, v) ;
printf("\n u = %d          v = %d ", u, v) ;
fct2(&u, &v) ;
printf("\n u = %d          v = %d ", u, v) ;
}
void fct1( int u, int v) {
u = u + 5 ; v= v - 1 ;
```

```
printf("\n u = %d            v = %d ", u, v) ;
}
void fct2( int *pu, int *pv)
{
*pu = *pu+1 ; *pv =*pv+2 ;
printf("\n u = %d   v = %d ", *pu, *pv) ;
}
```

Exercice 4

Que va afficher le programme suivant :

```
#include <stdio.h>
void fct1(int *,int *) ;
void fct2(int,int *) ;
void main()
{
int a = 4, b = 9, c = 3, d = 2 ;
fct1(&a, &b) ;
printf("\n a = %d  b = %d  c = %d  d = %d ", a, b, c, d);
fct2(a,&c) ;
printf("\n a = %d  b = %d  c = %d  d = %d ", a, b, c, d);
fct1(&b, &d) ;
printf("\n a = %d  b = %d  c = %d  d = %d ", a, b, c, d);
fct2(d,&b) ;
printf("\n a = %d  b = %d  c = %d  d = %d ", a, b, c, d);
}
void fct1(int *x,int *y)
{*x = *x + 2 ; *y = *y + 5 ;}
void fct2(int x,int *y)
{x = x + 3 ; *y = *y - 1 ;}
```

Exercice 5

Ecrire un programme qui permet de saisir, de trier par ordre décroissant et d'afficher un tableau de réels de dimension 5.

La saisie se fait dans une fonction qui retourne ce tableau au main(), qui le passe à son tour à une autre fonction qui le trie par ordre décroissant.

L'affichage du tableau trié se fera dans le main().

Corrigé du TP N° 7

Exercice 1

u = 5
*pu = 5
v = 7
*pv = 7

Exercice 2

i = 20
j = 37
*pj = 37
*pi = 37
*pi + 3 = 40

Exercice 3

u = 1 v = 3
u = 6 v = 2
u = 1 v = 3
*pu = 2 *pv = 5
u = 2 v = 5

Exercice 4

a = 6	b = 14	c = 3	d = 2
a = 6	b = 14	c = 2	d = 2
a = 6	b = 16	c = 2	d = 7
a = 6	b = 15	c = 2	d = 7

Exercice 5

```
#include <stdio.h>
#include <alloc.h>
float * Saisie() ;
void TriDecroissant(float t[]) ;
void main (void)
{
```

```
float *t ;   int i ;
t = Saisie() ;
TriDecroissant(t) ;
printf("\n Tableau trié : ") ;
for(i=0 ; i<5 ; ++i)
        printf("\n t[%d] = %.2f ", i, t[i]) ;
}

float * Saisie()
{
float *t=(float*)malloc(5*sizeof(float)) ;   int i ;
        for(i=0 ; i<5 ; ++i)
        {
        printf("\n Donnez t[%d] : ", i) ;
        scanf("%f", &t[i]) ;
        }
return t ;
}

void TriDecroissant(float t[])
{
int i, j ; float temp ;
for(i=0 ; i<4 ; ++i)
        for(j=i+1 ; j<5 ; ++j)
                if(t[i] < t[j])
                        {
                        temp=t[i] ;
                        t[i]=t[j] ;
                        t[j]=temp ;
                        }
}
```

Chapitre 8 : Les fichiers de données

Définition

Un fichier est un ensemble d'enregistrements de même type.

Exemple :
Fichier « client ». Chaque enregistrement de ce fichier contient les informations suivantes : numéro client, nom, adresse et téléphone.

Ouverture et fermeture d'un fichier de données

Avant de travailler sur un fichier (en lecture, ou en écriture, ou en lecture et écriture), il faut l'ouvrir avec la fonction fopen().

Syntaxe :
```
FILE   * variable ;
variable = fopen( <nom fichier>, "mode") ;
```

Où « variable » est un pointeur de type FILE (FILE écrit en entier en majuscules). <nom fichier> peut être une chaîne de caractères contenant le nom du fichier ou directement le nom du fichier placé entre guillemets.

Pour le mode d'ouverture du fichier, voir tableau ci-dessous.

De plus, tout fichier ouvert doit être fermé après usage avec la fonction fclose().

Syntaxe :
```
fclose(variable) ;
```

Les fonctions fopen() et fclose() sont contenues dans le fichier entête « stdio.h ».

Exemple :
Soit un fichier de données «clients.dat » déjà existant.

```
/* On déclare une variable pointeur x de type FILE */
FILE *x ;
```

```
/* On ouvre le fichier « clients.dat », par exemple, en lecture */
x = fopen("clients.dat","r") ;

/* On ferme le fichier après usage */
fclose(x) ;
```

Mode d'ouverture	Signification
"r"	Ouverture d'un fichier existant en lecture.
"w"	Création et ouverture d'un fichier en écriture seulement. Si le fichier existe déjà, il est détruit et remplacé par le fichier crée.
"a"	Ouverture d'un fichier existant pour lui ajouter des enregistrements (à la suite de ceux existants). Si le fichier n'existe pas au préalable, il est alors crée.
"r+"	Ouverture en lecture/écriture d'un fichier déjà existant.
"w+"	Création d'un fichier et ouverture en lecture/écriture. Si le fichier existe déjà, il est détruit et remplacé par le fichier crée.
"a+"	Ouverture en lecture/écriture d'un fichier auquel on peut ajouter des enregistrements (à la suite des enregistrements existants). Si le fichier n'existe pas, il est alors crée.

Si l'ouverture du fichier échoue (fichier inexistant, support de sauvegarde détérioré, ...), la fonction fopen() retourne un entier égal à zéro ou à NULL (NULL est une constante du C qui vaut zéro). Il est donc recommandé suite à l'ouverture d'un fichier de faire un test comme suit :

```
if ( x == NULL)    printf("\n L'ouverture du fichier a échoué ");
```

Si le message ne s'affiche pas, l'ouverture du fichier a réussi.

Lecture et écriture dans un fichier

On peut créer, consulter et modifier un fichier de données. Pour cela, on utilise les fonctions fread() et fwrite(). Ces deux fonctions sont contenues dans le fichier entête « stdio.h ».

La fonction fread() peut lire dans le fichier un ou plusieurs enregistrements à la fois.
La fonction fwrite() peut écrire dans le fichier un ou plusieurs enregistrements à la fois.

Syntaxe :

```
/* lecture d'un enregistrement dans le fichier */
fread(&variable, sizeof(<nom structure>), 1, x) ;
/* écriture d'un enregistrement dans le fichier */
fwrite(&variable, sizeof(<nom structure>), 1, x) ;
```

Où <nom structure> désigne la structure définissant le type d'enregistrements du fichier, « variable » est une variable de type <nom structure> et x le nom de la variable pointeur de type « FILE » vers laquelle on a ouvert le fichier.

« variable » peut être un nom de tableau. Dans ce cas, la syntaxe de lecture et écriture devient :

```
fread(variable, sizeof(<nom structure>), n, x) ;
fwrite(variable, sizeof(<nom structure>), n, x) ;
```

Où « n » désigne le nombre d'enregistrements lus dans le fichier ou écrits vers le fichier en une seule opération de lecture ou d'écriture.

Lorsqu'on modifie un fichier, on doit utiliser 2 variables pointeurs : un pointeur source et un pointeur cible. Le pointeur cible sert à éviter tout risque d'erreur. Dans ce cas, le fichier doit être ouvert deux fois en mode "r+" : une fois vers le pointeur source (pour la lecture) et une autre fois vers le pointeur cible (pour l'écriture).

Exemple :
Programme C qui permet de créer un fichier « etudiant.dat ». Chaque enregistrement de ce fichier contient les informations suivantes : Nom, Prénom et moyenne.

Le programme permet également d'afficher tout le fichier, et de modifier la moyenne d'un étudiant donné.

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct etudiant
{
char nom[20], prenom[20];
float moyenne ;
} ;
void main()
{
FILE *x, *y ;
etudiant E ;
char reponse, nom[20];
/* Création du fichier " etudiant.dat " */
x=fopen("etudiant.dat","w");
do      {
        clrscr() ;
        printf("Entrez les nom, prénom et la moyenne :") ;
        scanf("%s%s%f", E.nom, E.prenom, &E.moyenne) ;
        fwrite(&E, sizeof(etudiant), 1, x) ;
        printf("\n Autre saisie ? o/n") ;
        reponse=getche();
       }  while(reponse=='o') ;
fclose(x) ;
/* Consultation du fichier " etudiant.dat " */
clrscr() ;
x=fopen("etudiant.dat","r");
fread(&E, sizeof(etudiant), 1, x) ;
while( !feof(x)) /* tant que non fin de fichier */
       {
       printf("\n%s %s %f", E.nom, E.prenom, E.moyenne) ;
       fread(&E, sizeof(etudiant), 1, x) ;
       }
fclose(x) ;
getch();

/* Modification de la moyenne d'un étudiant */
clrscr() ;
printf("Nom de l'étudiant recherché : ") ;
```

```
scanf("%s", nom) ;
x=fopen("etudiant.dat","r+");
/* ouverture vers pointeur source x */
y=fopen("etudiant.dat","r+");
/* ouverture vers pointeur cible y */
fread(&E, sizeof(etudiant), 1, x) ;
while( !feof(x)) /* tant que non fin de fichier */
        {
        if(strcmp(nom, E.nom)==0)
                {
                printf("\n Nouvelle moyenne : ") ;
                scanf("%f", &E.moyenne) ;
                }
        fwrite(&E, sizeof(etudiant), 1, y) ;
        fread(&E, sizeof(etudiant), 1, x) ;
        }
fclose(x) ; fclose(y) ;
/* Consultation du fichier " etudiant.dat " après modification */
clrscr() ;
x=fopen("etudiant.dat","r");
fread(&E, sizeof(etudiant), 1, x) ;
while( !feof(x)) /* tant que non fin de fichier */
        {
        printf("\n%s %s %f", E.nom, E.prenom, E.moyenne) ;
        fread(&E, sizeof(etudiant), 1, x) ;
        }
fclose(x) ;
}
```

TP N° 8

Exercice 1

Ecrire un programme C qui permet :

1. de créer le fichier « employe.dat » avec les informations suivantes sur chaque employé : Nom, prénom et Adresse (numéro de rue, nom de rue, nom de ville et code postal).
2. de consulter le fichier « employe.dat ».
3. de modifier l'adresse d'un employé donné. La recherche de l'employé dans le fichier se fait par le nom et le prénom.

Exercice 2

Soit un fichier « etudiant.dat » contenant les noms, prénoms et les moyennes générales de tous les étudiants d'une école donnée.
Ecrire un programme C qui contient une fonction qui reçoit en paramètres le nom et le prénom d'un étudiant et retourne sa moyenne.
On suppose que le fichier « etudiant.dat » existe déjà et est sauvegardé sur disque.

Exercice 3

Soit un fichier « etudiant.dat », déjà existant sur disque, contenant les noms et prénoms de tous les étudiants d'une école donnée.
Ecrire un programme C qui contient une fonction qui reçoit en paramètre le nom d'un étudiant et retourne son prénom.

Corrigé du TP N° 8

Exercice 1

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct adresse
        { int no ; char rue[20], ville[20], cp[10];  } ;
typedef struct employe
        { char nom[20], prenom[20];  adresse adr ; } ;

int menu() ; /* fonction pour afficher le menu */
void main()
{
FILE *x, *y ;
employe E ;
char reponse, nom[20], prenom[20];
int choix ;
do {
  choix = menu() ;
  switch(choix)
        {
        /* -------------------------------------------*/
```

```
        case 1 : /* Création du fichier employe.dat */
        x=fopen("employes.dat","w");
        do {
         clrscr() ;
         printf("Nom et prénom de l'employé :") ;
         scanf("%s%s", E.nom, E.prenom) ;
         printf("\n Son adresse (No, rue, ville et cp):") ;
         scanf("%d%s%s%s", &E.adr.no, E.adr.rue, E.adr.ville, E.adr.cp) ;
         fwrite(&E, sizeof(employe), 1, x) ;
         printf("\n Autre saisie ? o/n") ;
         reponse=getche();
        }while(reponse=='o') ;
        fclose(x) ;
        break ;
/* ----------------------------------------------*/

        case 2 : /* Consultation du fichier employe.dat */
        clrscr() ;
        x=fopen("employes.dat","r");
        fread(&E, sizeof(employe), 1, x) ;
        while( !feof(x))
        {
        printf("\n%s %s ", E.nom, E.prenom) ;
        printf("%d %s %s %s", E.adr.no, E.adr.rue, E.adr.ville, E.adr.cp) ;
        fread(&E, sizeof(employe), 1, x) ;
        }
        fclose(x) ;
        getch() ;
        break ;
/* ----------------------------------------------*/

        case 3 :/* Modification de l'adresse */
        clrscr() ;
        printf("Nom et prénom de l'employé recherché : ") ;
        scanf("%s%s", nom, prenom) ;
        x=fopen("employes.dat","r+");
        y=fopen("employes.dat","r+");
        fread(&E, sizeof(employe), 1, x) ;
        while( !feof(x))
        {
        if(strcmp(nom,E.nom)==0 && strcmp(prenom,E.prenom)==0)
                {
```

```
                printf("\n Nouvelle adresse : ") ;
                scanf("%d%s%s%s", &E.adr.no, E.adr.rue,
                                        E.adr.ville, E.adr.cp) ;
                }
        fwrite(&E, sizeof(employe), 1, y) ;
        fread(&E, sizeof(employe), 1, x) ;
        }
        fclose(x) ; fclose(y) ;
/* ---------------------------------------------*/

}/* accolade du switch */
}while(choix != 4) ;
}

/* ---------------------------------------------*/
int menu()
{
int choix ;
clrscr() ;
printf("1. Création \n 2.Consultation") ;
printf("\n3.Modification \n4.Quitter l'application") ;
printf("\n Choix menu : ") ; scanf("%d", &choix) ;
return choix ;
}
```

Exercice 2

```
#include <stdio.h>
#include <string.h>
typedef struct etudiant
        { char nom[20], prenom[20];  float moyenne ; } ;

float recherche(char n[], char p[]) ;
void main()
{
char n[20], p[20] ;
float moy ;
printf("Nom et prénom de l'étudiant : ") ;
scanf("%s%s", n, p) ;
moy = recherche(n, p) ;
printf("\n Moyenne = %.2f", moy) ;
}
```

```
float recherche(char n[], char p[])
{
FILE *x ;
etudiant E ;
x=fopen("etudiant.dat","r");
fread(&E, sizeof(etudiant), 1, x) ;
while( !feof(x))
  {
  if(strcmp(n, E.nom)==0 && strcmp(p, E.prenom)==0)
                    break ;
  fread(&E, sizeof(etudiant), 1, x) ;
  }
fclose(x) ;
return E.moyenne ;
}
```

Exercice 3

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
typedef struct etudiant
          {
          char nom[20], prenom[20];
          } ;

char* recherche(char n[]) ;

void main()
{
char n[20], *p;
printf("Nom de l'étudiant : ") ;
scanf("%s", n) ;
p = recherche(n) ;
printf("\n Prénom : %s", p) ;
}

char* recherche(char n[])
{
char *p = (char*)malloc(20*sizeof(char)) ;
FILE *x ;
etudiant E ;
```

```
x=fopen("etudiant.dat","r");
fread(&E, sizeof(etudiant), 1, x) ;
while( !feof(x))
        {
        if(strcmp(n, E.nom)==0)
                { strcpy(p, E.prenom) ;
                  break ; }
        fread(&E, sizeof(etudiant), 1, x) ;
        }
fclose(x) ;
return p ;
}
```

Chapitre 9 : Les structures et les fonctions

Dans ce chapitre, nous étudierons les passages de variables de types structures à des fonctions. Il y a deux types de passage : par valeur et par adresse. Le passage par adresse a deux avantages par rapport au passage par valeur : rapidité d'exécution et économie d'espace mémoire.

Passage par valeur d'une structure à une fonction

Syntaxe :

Déclaration de la fonction :

```
<type> <nom fonction>(<nom structure>) ;
```

Appel de la fonction :

```
<nom fonction>(<variable>);
```

Définition de la fonction :

```
<type>  <nom fonction>(<nom structure> <variable>)
{
.................................
}
```

Où <variable> est une variable de type <nom structure>.

Exemple :

Programme C qui contient une fonction ajout() qui reçoit en paramètre une variable structure par valeur. On suppose que le fichier «produits.dat» existe déjà et est sauvegardé sur disque.

```
#include <stdio.h>
typedef struct produit
        { int no_produit ;  char nom_produit[20] ; };
/* déclaration de la fonction ajout */
void ajout(produit ) ;
void main()
{
produit prod ;
printf("\n No et nom du produit à ajouter : ") ;
scanf("%d%s", &prod.no_produit, prod.nom_produit) ;
ajout(prod) ; /* Passage par valeur */
}
```

```
/* définition de la fonction ajout */
void ajout(produit  p)
{
FILE *x ;
x=fopen("produits.dat", "a") ; /* ouverure du fichier en mode ajout */
fwrite(&p, sizeof(produit), 1, x) ;
fclose(x) ;
}
```

Retour par valeur d'une structure par une fonction

Syntaxe :

Déclaration de la fonction :

```
<nom structure>  <nom fonction>(paramètres) ;
```

Appel de la fonction :

```
<variable> = <nom fonction>(paramètres);
```

Définition de la fonction :

```
<nom structure>  <nom fonction>(paramètres)
{
................................
}
```

Où <variable> est une variable de type <nom structure>. Cette variable reçoit la structure de même type retournée par la fonction.

Exemple :
Programme C qui contient une fonction recherche() qui reçoit en paramètre un entier et retourne une variable structure par valeur. On suppose que le fichier «produits.dat» existe déjà et est sauvegardé sur disque.

```
#include <stdio.h>
#include <string.h>
typedef struct produit
{ int no_prod ; char nom_prod[20] ;
  float prix_prod;
};
/* déclaration de la fonction recherche */
produit  recherche(int ) ;
void main()
{ produit  p ;
```

```
int no;
printf(" \n No du produit à consulter : ") ;
scanf("%d", &no) ;
p = recherche(no) ; /* Appel de la fonction */

printf(" \n Voici les informations recherchées : ") ;
printf(" \n No produit : %d ", p.no_prod) ;
printf(" \n Nom produit : %s ", p.nom_prod) ;
printf(" \n Prix produit : %.2f ", p.prix_prod) ;
}
/* définition de la fonction recherche */
produit  recherche(int no)
{
produit p ;
FILE *x ;
x=fopen("produits.dat", "r") ;
fread(&p, sizeof(produit), 1, x) ;
while( !feof(x))
        {
        if(no == p.no_prod)
                break ;
        fread(&p, sizeof(produit), 1, x) ;
        }
fclose(x) ;  return  p ;
}
```

Passage par adresse d'une structure à une fonction

Syntaxe :

Déclaration de la fonction :

```
<type> <nom fonction>(<nom structure> *) ;
```

Appel de la fonction :

```
<nom fonction>(&<variable>);
```

Définition de la fonction :

```
<type> <nom fonction>(<nom structure>  *<pointeur>)
{
..............................
}
```

Où <variable> et <pointeur> sont respectivement une variable et un pointeur de type <nom structure>.

A l'appel de la fonction, <pointeur> reçoit l'adresse de <variable>. L'accés aux données membres de <variable> dans la fonction se fait par l'intermédiare de <pointeur> comme suit :

pointeur->membre

Le symbole « -> » est composé de deux caractères : - (moins) et > (supérieur).

Exemple :
Programme C qui contient une fonction ajout() qui reçoit en paramètre une variable structure par adresse. On suppose que le fichier «produits.dat» existe déjà et est sauvegardé sur disque.

```
#include <stdio.h>
typedef struct produit
        {
        int no_produit ; char nom_produit[20] ;
        } ;

/* déclaration de la fonction ajout */
void ajout(produit *) ;

void main()
{
produit prod ;
printf(" \n No et nom du produit à ajouter : ") ;
scanf("%d%s", &prod.no_produit, prod.nom_produit) ;
ajout(&prod) ; /* Passage par adresse */
}
/* définition de la fonction ajout */
void ajout(produit *p)
{
FILE *x ;
x=fopen("produits.dat", "a") ;
fwrite(p, sizeof(produit), 1, x) ;
fclose(x) ;
}
```

Remarquez que dans la fonction fwrite(), on a écrit « p » et non pas « &p » car p est un pointeur.

Retour par adresse d'une structure par une fonction

Syntaxe :

Déclaration de la fonction :

<nom structure> * <nom fonction>(paramètres) ;

Appel de la fonction :

<pointeur> = <nom fonction>(paramètres);

Définition de la fonction :

```
<nom structure>  *  <nom fonction>(paramètres)
{
................................
}
```

Où <pointeur> est un pointeur de type <nom structure>. Ce pointeur reçoit l'adresse de la structure de même type retournée par la fonction.

L'accés aux données membres de <pointeur> dans le main() ou dans la fonction appelante se fait comme suit :

pointeur->membre

Exemple :
Programme C qui contient une fonction recherche() qui reçoit en paramètre un entier et retourne une variable structure par adresse.
On suppose que le fichier «produits.dat» existe déjà et est sauvegardé sur disque.

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
typedef struct produit   { int no_prod ;   char nom_prod[20] ;
                           float prix_prod;
                         };
/* déclaration de la fonction recherche */
produit*  recherche(int ) ;
void main()
{
produit  *p ;   int no;
printf(" No du produit à consulter : ") ; scanf("%d", &no) ;
p = recherche(no) ; /* Appel de recherche */
printf(" \n Voici les informations recherchées : ") ;
printf(" \n No produit : %d ", p->no_prod) ;
```

```
printf(" \n Nom produit : %s ", p->nom_prod) ;
printf(" \n Prix produit : %.2f ", p->prix_prod) ;
free(p) ;
}
/* définition de la fonction recherche */
produit  * recherche(int no)
{
produit *p = (produit *)malloc(sizeof(produit)) ;
FILE *x ;
x=fopen("produits.dat", "r ") ; fread(p, sizeof(produit), 1, x) ;
while( !feof(x))
        {
        if(no == p->no_prod)  break ;
        fread(p, sizeof(produit), 1, x) ;
        }
fclose(x) ;
return  p ;
}
```

Remarquez, là aussi, que dans la fonction fread(), on a écrit « p » et non pas « &p » car p est un pointeur.

On a déjà dit, dans le chapitre sur les pointeurs, que lorsqu'une fonction se termine, le lien entre le reste du programme et les variables locales de cette fonction est coupé. Pour maintenir le lien avec l'emplacement mémoire contenant la variable dont on retourne l'adresse, il faut faire une allocation de mémoire en utilisant la fonction malloc().

Dans l'exemple précedent, la fonction malloc() alloue un espace mémoire de la taille de la structure « produit », et fournit en retour l'adresse de cet espace, qu'on assigne au pointeur p. Ce pointeur est retourné par la fonction au main() qui le reçoit dans un autre pointeur p, pour ensuite afficher les informations qui s'y trouvent.

Passage d'un tableau de structures à une fonction

Syntaxe :

Déclaration de la fonction :

```
<type> <nom fonction>(<nom structure> <nom tableau>[]) ;
```

Appel de la fonction :

```
<nom fonction>(<nom tableau>) ;
```

Définition de la fonction :

```
<type> <nom fonction>(<nom structure> <nom tableau>[])
  {
  ................................
  }
```

Exemple :
Soit la structure suivante :

```
typedef struct employe { char nom[20], prenom[20]; float salaire ;} ;
```

Le programme ci-dessous contient une fonction qui reçoit en paramètre un tableau de structures de type « employe », puis le trie par ordre décroissant des salaires. L'affichage du tableau ainsi trié se fait dans le programme principal.

```
#include <stdio.h>
#include <conio.h>
typedef struct employe
        {
        char nom[20], prenom[20] ; float salaire ;
        };

/* déclaration de la fonction tri */
void tri(employe  emp[]) ;

void main()
{
employe emp[10] ; int i ;
for(i=0 ;i<10 ;++i)
        {
        printf("\nEntrez nom, prénom et salaire : ") ;
        scanf("%s%s%f", emp[i].nom, emp[i].prenom, & emp[i].salaire) ;
        }
tri(emp) ; /* Passage du tableau de structure */
/* Affichage du tableau trié */
clrscr() ;
for(i=0 ;i<10 ;++i)
        printf("\n%s %s %f",emp[i].nom, emp[i].prenom, emp[i].salaire) ;
}
```

```
/* définition de la fonction tri */
void tri(employe emp[])
{
int i, j ; employe e ;
for(i=0 ;i<9 ;++i)
        for(j=i+1 ; j<10 ; ++j)
          if(emp[i].salaire < emp[j].salaire)
            { e= emp[i] ; emp[i]= emp[j] ; emp[j]=e ; }
}
```

Le passage d'un tableau de structures à une fonction, comme pour les autres types de tableaux, est un passage par adresse. Ainsi, les modifications subies par le tableau de structures dans la fonction apparaîssent dans la suite du programme.

TP N° 9

Exercice 1

Soit un fichier « clients.dat » déjà existant contenant les enregistrements suivants : Nom, prénom, adresse (No rue, rue, ville, code postal).

Ecrire un programme C qui contient une fonction qui reçoit en paramètre le nom et le prénom d'un client sous forme d'une structure (Passage par valeur) et retourne son adresse sous forme d'une structure (retour par valeur).

Exercice 2

Même énoncé que l'exercice 1 avec passage et retour par adresse.

Exercice 3

Soit une structure « etudiant » contenant les informations suivantes :
Numéro étudiant, nom et prénom.

Ecrire un programme C qui contient une fonction qui reçoit en paramètre un tableau de structures de type « etudiant », puis le trie par ordre alphabétique. L'affichage du tableau ainsi trié doit se faire dans le programme principal.

Corrigé du TP N° 9

Exercice 1

```
#include <stdio.h>
#include <string.h>
typedef struct adresse { int num ; char rue[20], ville[20], cp[10]; } ;
typedef struct client { char nom[20], prenom[20]; } ;
adresse recherche(client) ;
void main()
{ adresse A ; client C ;
printf("Nom et prénom du client : ") ; scanf("%s%s",C.nom, C.prenom) ;
A = recherche(C) ;
printf("\n Adresse : %d %s %s %s", A.num, A.rue, A.ville, A.cp) ;
}
adresse recherche(client C)
{
adresse Ad ; client Cl ;  FILE *x ; x=fopen("clients.dat","r");
fread(&Cl, sizeof(client), 1, x) ; fread(&Ad, sizeof(adresse), 1, x) ;
while( !feof(x))
  { if(strcmp(C.nom, Cl.nom)==0 && strcmp(C.prenom, Cl.prenom)==0)
                 break ;
    fread(&Cl, sizeof(client), 1, x) ;
    fread(&Ad, sizeof(adresse), 1, x) ;
  }
fclose(x) ;  return Ad ;
}
```

Exercice 2

```
#include <stdio.h>
#include <string.h>
#include <alloc.h>
typedef struct adresse { int num ; char rue[20], ville[20], cp[10]; } ;
typedef struct client  { char nom[20], prenom[20]; } ;
adresse* recherche(client*) ;
void main() { adresse *A ;  client C ;
printf("Nom et prénom du client : ") ;  scanf("%s%s", C.nom, C.prenom) ;
A = recherche(&C) ;
printf("\n Adresse : %d %s %s %s", A->num, A->rue, A->ville, A->cp) ;
}
```

```
adresse* recherche(client* C)
{
adresse *Ad = (adresse*)malloc(sizeof(adresse)) ; client Cl ;  FILE *x ;
x=fopen("clients.dat","r");
fread(&Cl, sizeof(client), 1, x) ; fread(Ad, sizeof(adresse), 1, x) ;
while( !feof(x))
 {
 if(strcmp(C->nom, Cl.nom)==0 && strcmp(C->prenom, Cl.prenom)==0)
                  break ;
 fread(&Cl, sizeof(client), 1, x) ; fread(Ad, sizeof(adresse), 1, x) ;
 }
fclose(x) ;  return Ad ;
}
```

Exercice 3

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
typedef struct etudiant { int no ; char nom[20], prenom[20]; } ;
void TriAlphabetique(etudiant E[]) ;
void main()
{  etudiant E[10] ; int i ;
  for(i=0 ; i<10 ; ++i)
     {   clrscr() ;
         printf("No, nom et prénom de l'étudiant : ") ;
         scanf("%d%s%s", &E[i].no, E[i].nom, E[i].prenom) ;
     }
  TriAlphabetique(E) ;
  clrscr() ;
  for(i=0 ; i<10 ; ++i)
        printf("\n%d %s %s", E[i].no, E[i].nom, E[i].prenom);
}
void TriAlphabetique(etudiant E[])
{  int i,j ; etudiant temp ;
  for(i=0 ; i<9 ; ++i)
   for(j=i+1 ; j<10 ; ++j)
     if(strcmp(E[i].nom, E[j].nom) > 0)
             { temp = E[i] ; E[i] = E[j] ; E[j] = temp ; }
}
```

Chapitre 10 : Les listes chaînées

Définition

Une liste chaînée est un ensemble d'éléments d'informations appelés noeuds. En plus de l'information dont il est porteur, un noeud possède un pointeur. Ce pointeur contient l'adresse du noeud suivant dans la liste.

L'adresse de début de la liste chaînée (l'adresse du premier nœud) doit être sauvegardée dans un pointeur à part.

Syntaxe :
```
typedef struct <nom structure> {
                    <membre 1> ;
                    ..............
                    <membre k> ;
                    <nom structure> *<pointeur>;
                    };
```

Où <pointeur> est un pointeur de type <nom structure> contenant l'adresse de l'élément suivant de la liste chaînée.

Exemple :
```
typedef struct vehicule
        {
          char nom_vehicule[20] ;
          float prix ;
          vehicule *suivant ;
        };
```

« nom_vehicule » et « prix » représentent l'information dont le noeud est porteur, et « suivant » est le pointeur qui contient l'adresse du noeud suivant dans la liste.

Gestion dynamique de la mémoire

Une liste chaînée n'est pas limitée à un nombre fixe d'éléments. Elle peut être facilement étendue ou diminuée selon les besoins. C'est la gestion dynamique de la mémoire. Ceci est possible grâce aux fonctions malloc() et free() contenues dans le fichier entête « alloc.h ».

La gestion dynamique de la mémoire permet d'utiliser uniquement l'espace mémoire nécessaire à l'exécution d'un programme. Ce qui n'est pas le cas, lorsqu'on utilise par exemple, un tableau de structures, car il doit être déclaré avec une dimension fixe.
On rappelle que :
- La fonction malloc() alloue un espace mémoire de la taille de la structure en octets, et fournit l'adresse de cet espace en retour (cette adresse est assignée à un pointeur de même type que la structure).
- La fonction free() permet de libérer un emplacement préalablement alloué lorsqu'on en a plus besoin.

Les listes chaînées simples

Une fois une liste chaînée créée, on peut la consulter bien sûr, mais aussi insérer et supprimer des éléments sans restrictions. Ce qui est l'intérêt premier des listes chaînées.

A noter qu'on peut insérer et supprimer un élément n'importe où dans la liste (au début, à la fin, ou à n'importe quelle position).

Exemple :
Programme C qui permet de créer une liste chaînée simple, de la consulter, d'insérer un élément au début de la liste et de supprimer le premier élément de la liste.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
typedef struct vehicule
        { char nom_vehicule[20] ; float prix ;  vehicule *suivant ;       };
void main()
{
vehicule *debut, *courant , *nouveau;   char choix ;

/* Création de la liste chaînée */
/* On sauvegarde l'adresse du 1er nœud dans le pointeur " debut " */
debut = (vehicule*)malloc(sizeof(vehicule)) ;  courant = debut ;
do {    clrscr() ;
        printf("\n Entrez nom véhicule : " ) ;
        scanf("%s", courant->nom_vehicule) ;
```

```
        printf("\n Entrez prix véhicule : " ) ;
        scanf("%f", &courant->prix) ;
        printf("\n Autre saisie ? o/n " ) ;
        choix = getche() ;
        if(choix == 'o' || choix == 'O')
                {
                courant->suivant=(vehicule*)malloc(sizeof(vehicule));
                courant = courant->suivant ;
                }
        else    courant->suivant = NULL ;
    } while(choix == 'o' || choix == 'O') ;
/* Consultation de la liste chaînée */
clrscr() ; courant = debut ;
while(courant != NULL)
 {
 printf("\n %s %.2f",courant->nom_vehicule, courant->prix) ;
 courant= courant->suivant ;
 }

/* Ajout d'un élément au début de la liste */
nouveau = (vehicule*)malloc(sizeof(vehicule)) ;
printf("\nEntrez nom véhicule : " ) ; scanf("%s",nouveau->nom_vehicule) ;
printf("\n Entrez prix véhicule : " ) ;  scanf("%f", &nouveau->prix) ;
nouveau->suivant = debut ;  debut = nouveau ;
 /* Consultation de la liste chaînée */
clrscr() ; courant = debut ;
while(courant != NULL)
 {
 printf("\n %s %.2f",courant->nom_vehicule, courant->prix) ;
 courant= courant->suivant ;
 }  getch();

/* Supprimer un élément au début de la liste */
free(debut) ; debut = debut->suivant ;
/* Consultation de la liste chaînée */
clrscr() ; courant = debut ;
while(courant != NULL)
 {
 printf("\n %s %.2f",courant->nom_vehicule, courant->prix) ;
 courant= courant->suivant ;
 }
}
```

Les listes chaînées bidirectionnelles

Une liste chaînée bidirectionnelle est une liste chaînée dans laquelle chaque élément possède l'adresse de l'élément précédent et l'adresse de l'élément suivant. A noter, là aussi, qu'on peut insérer et supprimer un élément n'importe où dans la liste.

Une liste chaînée bidirectionnelle est plus facile à manipuler, lors des opérations d'insertion et de suppression d'éléments, qu'une liste chaînée simple.

Exemple :
Programme C qui permet de créer une liste chaînée bidirectionnelle, de la consulter, d'insérer un élément au début de la liste et de supprimer le premier élément de la liste.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
typedef struct vehicule
{ char nom_vehicule[20] ; float prix ;   vehicule *precedent, *suivant ; };
void main()
{
vehicule *debut, *courant, *nouveau ;
char choix ;
debut = (vehicule*)malloc(sizeof(vehicule)) ;
debut->precedent = NULL ; courant = debut ;
do
{    clrscr() ;
     printf("\n Entrez nom véhicule : " ) ;
     scanf("%s", courant->nom_vehicule) ;
     printf("\n Entrez prix véhicule : " ) ;
     scanf("%f", &courant->prix) ;
     printf("\n Autre saisie ? o/n " ) ;
     choix = getche() ;
     if(choix == 'o' || choix == 'O')
          {
          courant->suivant=(vehicule*)malloc(sizeof(vehicule));
          courant->suivant ->precedent = courant ;
          courant = courant->suivant ;
          }
```

```
        else courant->suivant = NULL ;
} while(choix == 'o' || choix == 'O') ;
/* Consultation de la liste chaînée */
clrscr() ;
courant = debut ;
while(courant != NULL)
 {
 printf("\n %s %.2f",courant->nom_vehicule, courant->prix) ;
 courant= courant->suivant ;
 }

/* Ajout d'un élément au début de la liste */
nouveau = (vehicule *)malloc(sizeof(vehicule)) ;
printf("\n Entrez nom véhicule : " ) ;
scanf("%s", nouveau->nom_vehicule) ;
printf("\n Entrez prix véhicule : " ) ;
scanf("%f", &nouveau->prix) ;
nouveau->precedent = NULL ;
nouveau->suivant = debut ;
debut = nouveau ;
 /* Consultation de la liste chaînée */
clrscr() ;
courant = debut ;
while(courant != NULL)
 {
 printf("\n %s %.2f",courant->nom_vehicule, courant->prix) ;
 courant= courant->suivant ;
 }getch();

/* Supprimer un élément au début de la liste */
free(debut) ;
debut = debut->suivant ;
debut->precedent = NULL ;
/* Consultation de la liste chaînée */
clrscr() ;
courant = debut ;
while(courant != NULL)
 {
 printf("\n %s %.2f",courant->nom_vehicule, courant->prix) ;
 courant= courant->suivant ;
 }
}
```

TP N° 10

Exercice 1

Soit la structure suivante :

```
typedef struct employe {
        int numero; char nom[20] ; float salaire ;
        employe *suivant ;
        };
```

Ecrire un programme C qui permet :

- de créer une liste chaînée avec la structure «employe»,
- de consulter la liste chaînée,
- d'ajouter un employé à n'importe quelle position de la liste chaînée (au début, à la fin, ou au milieu), et
- de supprimer un employé de n'importe quelle position de la liste chaînée (au début, à la fin, ou au milieu).

Exercice 2

Refaire l'exercice 1 en utilisant une liste chaînée bidirectionnelle. Ajouter pour cela, le pointeur « precedent » de type « employe » dans la structure ci-dessus.

Corrigé du TP N° 10

Exercice 1

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
typedef struct employe
     { int numero ; char nom[20] ; float salaire;  employe *suivant ; } ;
int MenuPrincipal() ;
int SousMenu() ;
void main()
{
employe *debut, *courant, *nouveau, *precedent ;
char reponse ;   int choix, SousChoix, num ;
do {
choix = MenuPrincipal() ;
```

```
switch(choix)
{
/* ------------------------------------------------*/
case 1 : /* Création de la liste chaînée */
debut = (employe *)malloc(sizeof(employe)) ;
courant = debut ;
do {
clrscr() ;
printf("No, nom et salaire de l'employé :") ;
scanf("%d%s%f", &courant->numero, courant->nom, &courant->salaire) ;
printf("\n Autre saisie ? o/n " ) ;  reponse = getche() ;
if(reponse == 'o' || reponse == 'O')
  {
  courant->suivant=(employe*)malloc(sizeof(employe));
  courant = courant->suivant ;
  }
else  courant->suivant = NULL ;
} while(reponse == 'o'  || reponse == 'O') ;
break ;
/* ------------------------------------------------*/
case 2 : /* Consultation de la liste chaînée */
clrscr() ;
courant = debut ;
while(courant != NULL)
 {
 printf("\n %d  %s %.2f", courant->numero,
                              courant->nom, courant->salaire) ;
 courant = courant->suivant ;
 }
getch() ;
break ;
/* ------------------------------------------------*/
case 3 :/* Ajout d'un élément à la liste */
clrscr() ;
nouveau = (employe *)malloc(sizeof(employe)) ;
printf("No, nom et salaire de l'employé :") ;
scanf("%d%s%f",&nouveau->numero,nouveau->nom,
                                          &nouveau ->salaire) ;
SousChoix = SousMenu() ;
switch(SousChoix)
                {
                case 1 : /* Ajout au début */
```

```
                nouveau->suivant = debut ;
                debut = nouveau ;
                break ;

                case 2 : /* Ajout à la fin */
                /* On se positionne sur le dernier élément de la liste :  */
                /* celui dont le suivant est égal à NULL */
                courant = debut ;
                while(courant->suivant != NULL)
                        courant = courant->suivant ;
                /* A la sortie de la boucle while, courant pointe sur le
                dernier  élément de la liste */
                nouveau->suivant = NULL ;
                courant->suivant = nouveau ;
                break ;

                case 3 : /* Ajout au milieu */
                clrscr() ;
                printf("No employé après lequel ajouter ?") ;
                scanf("%d", &num) ;
                courant = debut ;
                while(courant != NULL)   {
                        if(num == courant->numero)
                        {
                        nouveau->suivant = courant->suivant ;
                        courant->suivant = nouveau ;
                        break ; /* On sort de la boucle */
                        }
                        courant = courant->suivant ; }
                break ;
}/* fin du switch(SousChoix) */
break ;
/* ------------------------------------------------*/

case 4 : /* Suppression d'un élément de la liste */
SousChoix = SousMenu() ;
switch(SousChoix)
                {
                case 1 : /* Suppression au début */
                free(debut) ;
                debut = debut->suivant ;
                break ;
```

```
                case 2 : /* Suppression à la fin */
                /*On se positionne sur l'avant dernier élément de la liste :
                  celui dont le suivant du suivant est égal à NULL */
                courant = debut ;
                while(courant->suivant->suivant != NULL)
                        courant = courant->suivant ;
                /* A la sortie de la boucle while, courant pointe sur l'avant
                                                    dernier élément*/
                free(courant->suivant) ; /* On libère le suivant de l'avant
                          dernier, donc le dernier */
                courant->suivant = NULL ;
                break ;

                case 3 : /* Suppression au milieu */
                clrscr() ;
                printf("No employé à supprimer de la liste ?") ;
                scanf("%d", &num) ;
                precedent = debut ;
                courant = debut->suivant ; /* On commence la recherche
                        à partir du 2nd élément, car le cas du 1er élément
                        a déjà été traité */
                while(courant != NULL)
                        if(num == courant->numero)
                        {
                        free(courant) ;
                        precedent->suivant = courant->suivant ;
                        break ; /* On sort de la boucle */
                        }
                        else
                        {
                        precedent = courant ;
                        courant = courant->suivant ;
                        }
                break ;
}/* fin du switch(SousChoix) */
break ;

/* ----------------------------------------------------*/

}/* fin du switch(choix) */
}while(choix != 5) ;
}
```

```
/* ---------- Définition des fonctions -----------*/
int MenuPrincipal()
{ int choix ;  clrscr() ;
printf("1.Création \n 2.Consultation \n 3.Ajout");
printf("\n 4.Suppression \n 5.Quitter l'application");
printf("\n Choix menu : ") ; scanf("%d", &choix) ;
return choix ;
}
int SousMenu()
{ int choix ; clrscr() ;
printf("1. Au début \n 2. A la fin \n 3. Autre ") ;
printf("\n 4. Retour au menu principal ") ;
printf("\n Choix menu : ") ; scanf("%d", &choix) ;
return choix ;
}
```

Exercice 2

La structure de ce programme est identique à celle de l'exercice 1. Ce qui fait que nous ne reproduirons que l'essentiel. D'autre part, le principe de création et de consultation avant d'une liste chaînée bidirectionnelle ont déjà été traités dans un exemple ci-dessus.

```
/* Consultation arrière de la liste chaînée */
/* On commence par la fin */
courant = fin ;
while(courant != NULL)
  {
  printf("\n%d %s %.2f", courant->numero, courant->nom,
                                                    courant->salaire) ;
  courant = courant->precedent ;
  }

/* Ajout d'un élément à la liste */
/* Ajout au début */
nouveau->suivant = debut ;
nouveau->precedent = NULL ;
debut->precedent = nouveau;
debut = nouveau ;

/* Ajout à la fin */
fin->suivant = nouveau ;
```

```
nouveau->suivant = NULL ;
nouveau->precedent = fin ;
fin = nouveau ;

/* Ajout au milieu */
/* Une fois trouvé le numéro après lequel insérer */
nouveau->precedent = courant ;
nouveau->suivant = courant->suivant ;
courant->suivant->precedent = nouveau ;
courant->suivant = nouveau ;

/* Suppression d'un élément de la liste */
/* Suppression au début */
free(debut) ;
debut = debut->suivant ;
debut->precedent = NULL ;

/* Suppression à la fin */
free(fin) ;
fin = fin->precedent ;
fin->suivant = NULL ;

/* Suppression au milieu */
printf("No employé à supprimer de la liste ?") ;
scanf("%d", &num) ;
/* On fait une recherche avant et on commence à   partir du
                    2nd élémént */
courant = debut->suivant ;
while(courant->suivant != NULL)
                    if(num == courant->numero)
                   {
                   free(courant) ;
                   courant->precedent->suivant = courant->suivant ;
                   courant->suivant->precedent = courant->precedent ;
                   break ; /* On sort de la boucle */
                   }
                   else courant = courant->suivant ;
```

LE LANGAGE C

S.Graïne

Le langage C est l'un des langages de programmation les plus utilisés dans le monde. Il doit sa célébrité à sa puissance de calcul et à sa facilité relative de mise en œuvre.

Ce livre s'adresse à tous ceux qui souhaitent apprendre à programmer en langage C, ou tout simplement améliorer leurs connaissances de ce langage. Il est conçu de façon progressive : des notions les plus simples vers celles plus avancées. Il est pour cela constitué de 10 chapitres : Concepts de base du langage C, les structures de contrôle, les fonctions, les tableaux, les chaînes de caractères, les structures, les pointeurs, les fichiers de données, les structures et les fonctions, et enfin, les listes chaînées.

Chaque chapitre se termine par un travail pratique composé de quelques exercices. Tous les travaux pratiques sont entièrement corrigés. Ils constituent ainsi un supplément d'exemples appréciable.



ISBN 9961 723 26 0

Prix 190 DA